CA Repository for z/OS

Administration Guide

r7.2

CA Product References

This document references the following CA product:

- CA Repository for z/OS

Contact CA

Contact Technical Support

For your convenience, CA provides one site where you can access the information you need for your Home Office, Small Business, and Enterprise CA products. At http://ca.com/support, you can access the following:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Provide Feedback

If you have comments or questions about CA product documentation, you can send a message to techpubs@ca.com.

If you would like to provide feedback about CA product documentation, complete our short customer survey, which is also available on the CA Support website, found at http://ca.com/docs.

Contents

Chapter 1: Introduction 15

What You Need to Know 16
Repository Access and Support 16
User Access and TSO 16
Size Requirements 16
Multiple Subsystem Support 16
Internal and Third Party Tool Support 17
Set Up the Repository 17
Customize the Repository Using Control Tables and the Extend Model 17
Repository Extension 19
Guidelines for Extending 19
Types of Extensions 19
Metadata Security 20
Populate the Repository with Metadata 21
Maintain Repository Metadata 22
Logical and Physical Object Definition Support 22
Name Objects and Supporting Standards 22
Repository Maintenance 24

Chapter 2: The Extend Model 25

The Extend Model 25
Extend Model Object Types 25
The Extend Dialogs 32
Extend Commands 33
The BUILD Command 34
The IMPORT Command 37
The REMOVE Command 39
The VIEWMSG Command 40
The CREATE Command 40
The RECREATE Command 41
The NEXTID Command 42
The GRANT Command 43
The REVOKE Command 44

Chapter 3: Manage Repository Users 45

The User-Privilege Model-USERPRIV 45

Add a New User 46
Insert a USER Instance that Defines a New User into the Repository 46
Associate a User with Repository Privileges 47
Add a User Definition to the Control Tables 49
Modify a User Definition 49
Extract User Data from the Control Tables 50
Remove a User 52

Chapter 4: Working with Privileges 53

The Privilege Model 53
How to Add Privileges 55
Insert (Define) a Privilege Instance 55
Link the Privilege to Commands 56
Link the Privilege to Workstations 57
Link the Privilege to Dialogs 58
Limit the Privilege to Certain Entity Types 59
Limit a Privilege to Certain Statuses 60
Add Privileges to the Control Tables 61
Modify a Privilege 61
Extract Altered Privilege Data 62
Remove a Privilege 63
How to Set Up Default Privileges for New Users 63
Define Default Privileges 64
Activate the Default Privilege Feature 64
Command Checklist 65

Chapter 5: Working with Statuses 71

Repository Status and Level Numbers 71
Status Level Numbers 71
Level Number Restrictions 72
Migration Paths 73
The Status Model 73
Add a New Status to the Repository 75
Validate Statuses 76
Add a Migration Path 76
Limit the Migration Path to Specific Entity Types 77
Provide Migration Path Help Messages 78
Add a Migration Path to Control Tables 79
Modify Statuses and Migration Paths 80
Extract Data from Control Tables to Extend Data Tables 80
Remove a Status or Migration Path 82

Statuses That Support Projects . . . 83
Production Status (PROD) . . . 83
Test Status . . . 83
Development Status (DEVnn) . . . 84

Chapter 6: Working with Commands 85

The Command Model . . . 85
Define, Link, and Add a Command . . . 86
Define a Command Instance . . . 87
Provide Help for a Command . . . 89
Limit a Command to Specific Entity Types . . . 89
Add a Command to the Control Tables . . . 90
Modify a Repository Command . . . 91
Extract Command Data from the Control Tables . . . 91
Remove a Repository Command . . . 92

Chapter 7: Adding and Customizing Maps and Entity Types 93

Repository Entity Types . . . 93
The Quick Map Facility . . . 93
Use TSO Edit Commands . . . 94
Open the Quick Map Facility . . . 94
Use the Quick Map Facility . . . 96
Add Screen Literals . . . 96
Customize Characters Used for Input Fields . . . 99
Define Attribute Details . . . 100
Create On-Screen Links . . . 104
Example of Creating On-Screen Links . . . 106
Specify Storage Tables for Maps . . . 107
Link the Map to an Entity Type . . . 108
Link Maps to a Dialog . . . 110
Calculate Map Offsets . . . 111
Recreate an Existing DB2 Storage Table . . . 111
Save Maps . . . 112
Build History Objects . . . 112
Remove History Objects . . . 113
Setting Approval for Objects . . . 113

Chapter 8: Working with Code Tables 115

What Is a Code Table? . . . 115
The Code Table Model . . . 116
How You Create a Code Table . . . 117

Define a Code Table Instance . . . 117
Define a Variance . . . 119
Add Variances to a Code Table . . . 120
Provide Help for a Code Table . . . 121
Add a Code Table to the Control Tables . . . 122
Modify a Code Table . . . 123
Extract Code Table Data from the Control Tables . . . 123
Remove a Code Table . . . 124

Chapter 9: Working with Domains 125

Domain Keys and Values . . . 125
Domain Sets . . . 125
Domain Tables . . . 127
Set Up a Domain Dialog and Domain Tables . . . 128

Chapter 10: Customizing Messages 131

The CA Repository Messages . . . 131
Types of CA Repository Messages . . . 131
The Message Model . . . 133
Create a Help Message . . . 133
Insert a MESSAGE Instance . . . 134
Add Message Text to a Message . . . 136
Add Supplemental Messages . . . 137
Add a Message to the Control Tables . . . 138
Customize Existing Messages . . . 138
Extract Code Table Data from the Control Tables . . . 139
Customize the LOGON Message . . . 141
Remove a Message . . . 142

Chapter 11: Working with Keywords 143

Keywords . . . 143
Add a Keyword . . . 144
Modify a Keyword . . . 145
Import Data from Control Tables to Storage Tables . . . 145
Remove a Keyword . . . 147
Extend Keyword Reference Tables . . . 147

Chapter 12: Working with the Repository I/O Module 149

The I/O Module . . . 149
What Is the Repository I/O Module? . . . 149

Why Use the Repository I/O Module? 150
What Do You Use I/O Maps For? 150
What Do I/O Maps Contain? 150
The I/O Module Calling Pattern 150
The IO_VARS Variable 151
The BLOCK Variable 154
The WHERE_DATA Variable 155
The ORDER_BY Variable 156
The MAPAREA Variable 156
The SQLAREA Variable 157
The REASON Variable 157
The MSG_ID Variable 158
The RETURN_CODE Variable 158
The MSG_QUAL Variable 159
Special Considerations 159
I/O Module Maps 159
Open-Close-Fetch Actions 160
Message Processing 160
Map Selection and Performance Impact 160
SQLAREA Size 160
Decimal Fields 161
The Where Clause on SELECT or OPEN 161
Special "X" (Excluded) Attributes 161
UPDATE Statements 162
DELETE Statements 162
INSERT Statements 162
REPLACE Statements 163
FETCH 163
COMMIT and ROLLBACK 163
LOGGING 164
EXIT Routine 164
Entity Sets 164
Extended Text 165
Policies 165
Repository Control Tables 165
Sample Calls to the I/O Module 166
Sample Map Definition 166
Sample Select for a Known Entity ID 167
Sample Select for a Known Map (1 of 2) 168
Sample SELECT Using a Known Map (2 of 2) 169
Stored Procedure Example 170

Chapter 13: CA Repository z/OS or APIs 173

CGETVER . . . 174
Input . . . 174
Output . . . 175
How the Program Determines the Available Version Number . . . 175
CGETVER Example . . . 176
CMSGS . . . 177
Input . . . 177
InOut . . . 177
Output . . . 178
CMSGS Example . . . 178
CRTXT80 . . . 178
Input . . . 178
Input-Output . . . 181
Output . . . 181
The Global Temporary Table . . . 182
CRTXT80 Example . . . 183
DBXENTID . . . 184
Input . . . 184
Output . . . 185
DBXENTID Example . . . 185
DBXIA . . . 186
Input . . . 187
Output . . . 187
DBXIA Example . . . 188
DBXIOD . . . 189
Input . . . 189
Output . . . 189
DBXIOI . . . 190
Parameters . . . 191
DBXIOI Example . . . 194
DBXIOM . . . 195
Input . . . 196
Output . . . 196
DBXIOM Example . . . 197
DBXIOR . . . 198
Input . . . 198
Output . . . 198
DBXIOR Example . . . 199
DBXIOS . . . 200
Input . . . 201
Output . . . 202

DBXIOS Example 204
DBXIOU 211
Input 212
Output 214
DBXIOU Example 216
DBXNSVI 217
Input 217
Output 218
DBXNSVI Example 219
DBXSEC 219
Parameters 220
DBXSEC Example 221
Get_streamtext_32K 222
Input 223
Output 223
GET_STREAMTEXT_32k Example 224

Chapter 14: The Batch Load Facility 227

The Batch Load Statement 227
Repository LOAD Syntax for INSERT, UPDATE, DELETE, REUSE, and SELECT 229
Required Batch Load Syntax Parameters 230
Optional Batch Load Syntax Parameters 232
Destination Parameters 237
Field Declarations 237
Sample Batch Load Syntax 242
Additional Notes 242
Batch Load Syntax Scenarios 243
Batch INSERT: Load Metadata 244
Load Metadata into Associations and Relationships 245
Batch UPDATE: Change Metadata 246
Batch REUSE: Combine Inserts and Updates 247
Batch SELECT: Export Metadata 248
Batch DELETE: Delet Metadata 249
Workstation Processing with Batch Load 250
Delete Within Workstations 250
CA Repository-Supplied Examples of the Batch Load Facility 251
Sample JCL 251
Text Load 253
Text Load Facility JCL 254

Chapter 15: Backup, Recovery, and Performance Tuning 257

Control Tables and DB2 Tables . . . 257
Backup and Recovery Procedures . . . 257
What Components Must Be Backed Up? . . . 257
Tune the Database . . . 258

Chapter 16: History Tables 259

Enable the History Feature . . . 259
History Core Tables . . . 260
History Trail Processing-Special Considerations . . . 260
Product-Level History File Activation . . . 261
Metadata History Object Selection . . . 263
General Notes on Metadata History File Selection . . . 264

Chapter 17: Migration Extensions 267

PRMMODL Utility . . . 267
PRMMODL Utility Parameters . . . 269
Migrate User Updates Utility . . . 271
ADD . . . 271
REPLACE . . . 272
COPY . . . 272

Chapter 18: Creating CA Repository Paths 277

Path Report Facility . . . 277
Execute the Path Report Facility . . . 278
Path Delete Facility . . . 278
Execute the Path Delete Facility . . . 279
Run the Path Delete Facility in Report Mode . . . 279
Path Workstation Add Facility . . . 279
Objects Added to the Workstation . . . 280
Execute the Path Workstation Add Facility . . . 280
Run the Path Add Facility in Report Mode . . . 280
How Paths Are Processed . . . 281
Load Paths . . . 281
Add or Modify the Paths Definitions . . . 281
Create Path Definition Statements . . . 282
Example of a Path Definition Statement for a Path Report . . . 284
Path Report Processing . . . 285
Example of a Path Delete . . . 286
Delete Path Processing . . . 287

Create Path Attribute Statements . . . 287
Example of a Path Attribute Statement for a Path Report . . . 288
Specify the Path Facility Parameters . . . 289
Run-Time Parameters for the SYSIN Data Card . . . 290
Path Facilities Usage Notes . . . 291

Chapter 19: Using the Relational Translator 293

The Relational Translator . . . 293
The Relational Transfer Process . . . 294
Tie-Back Cross Referencing . . . 295
Workstation Processing . . . 295
Entity and Attribute Name Generation . . . 296
Element Re-Usability . . . 296
Open the Relational Translator Screen . . . 296
View the Results of Processing . . . 299
The Work Tables . . . 299
The Entity Control Table . . . 300
The Attribute Control Table . . . 300
Model Requirements . . . 302
The Logical Dialog Model to Create DBMS Component . . . 303
Control Table Layouts . . . 304
DBX_REL_TRAN_ATTR . . . 304
DBX_REL_TRAN_ENT . . . 305

Chapter 20: Working with User Exits 307

User Exits . . . 307
Add User Exits . . . 307
Writing an Exit . . . 309
List of ISPF Variables Available in User Exits (Shared Pool) . . . 310
Variables Put Out by Repository for Each Entity Before Executing the Exit . . . 311
Variables Retrieved by Repository After Executing the Exit . . . 312
Install User Exits . . . 313
Restore User Exits . . . 318

Chapter 21: Approval Process 319

Enable the Approval Feature . . . 319
Activate an Approval Process . . . 320
Add Approval for an Object . . . 320
Stop Approval Processing for an Object . . . 321
Setting Up Approvers . . . 321
Approving an Object . . . 322

Requests Validation Process 322
Approval Table 322

Appendix A: Utilities 325

PRMORPH Utility 325
Run PRMORPH 326
Example of PRMORPH Results 326
Access PRMORPH 327
PRMSTAT Utility 328
Run PRMSTAT 328
Access PRMSTAT 329
RRSPB Utility 330
DB2 Setting 331
Run RRSPB 333
Access RRSPB 334

Index 337

Chapter 1: Introduction

This guide serves as a reference for those who set up and maintain the CA Repository for z/OS. It is written for basic as well as advanced users. It explains how to:

- Implement and customize the repository
- Support and maintain the repository
- Use the Extend Model and commands
- Manage users
- Understand Statuses
- Define and maintain repository privileges
- Create and customize object types
- Define commands and add and customize messages
- Set up code tables
- Create and work with domains
- Add and update product keywords
- Plan, execute, and implement static maps
- Use the repository maps to manipulate data
- Use the batch load facility
- Perform backup and recover functions
- Define repository paths
- Create physical DB2 model definitions using the Relational Translator
- Work with User Exits
- Maintain the DB2 objects required to support the repository

This section contains the following topics:

What You Need to Know (see page 16)
Repository Access and Support (see page 16)
Set Up the Repository (see page 17)
Repository Extension (see page 19)
Repository Maintenance (see page 24)

What You Need to Know

To understand most of the concepts and procedures in this guide, you must be familiar with:

- z/OS
- DB2
- ISPF
- JCL

Some of the screens, messages, and pull-down menus shown in this guide may vary slightly from those at your site.

Repository Access and Support

This section briefly discusses issues relating to repository access, multiple system support, and tool support.

User Access and TSO

The CA Repository for z/OS runs as an ISPF task under TSO. You use a standard ISPF facility to operate the repository screens.

Size Requirements

You must be allocated a TSO region size of 6 MB below the line and 8 to 10 MB above the line to get full power for repository screens.

Multiple Subsystem Support

CA Repository for z/OS is a production application that supports development and uses the DB2 Call Attach Facility to enable you to:

- Store data in a single subsystem
- Act on any subsystem that can be attached through the DB2 Call Attach Facility

For example, the DB2 Catalog Synchronization Facility running from your repository in a test subsystem can read metadata out of the catalog of a production subsystem if the subsystem is specified on the DB2 CatSync panel.

Internal and Third Party Tool Support

CA Repository for z/OS has facilities that you use to execute internally developed or third-party vendor tools operating under TSO. You can build interfaces to these tools at the entity (or entity attribute) level. For example, you can interface with a user-written programmer's workbench directly from within the repository.

For specific information about interfaces to third party tools, contact CA Technical Support.

Set Up the Repository

Your first task is to determine which roles the repository will play within your organization. There are several ways you can use the repository, including the following:

Central Repository

You can use the repository as a central DB2-based repository to support databases including DB2, IMS, and logical data definitions. Using the repository's EXTEND dialog, you can customize the repository to meet your site's needs.

Data Dictionary

The base repository provides full DB2 and copybook support. Additionally, IMS support can be added providing a DB2-based, CUA compliant interface for performing functions supported by IBM's Data Dictionary (IMS) and the DSIMS Dictionary.

User Glossary

The repository provides a sophisticated yet easy-to-use dialog to help you understand and locate logical and physical data structures.

CASE Facilitator

CA Repository for z/OS includes interfaces to various CASE tools. You can use these interfaces to maintain a central and single point of metadata management across multiple tools and platforms. In this role, the repository can be a central library from which metadata can be checked out to support a project.

Customize the Repository Using Control Tables and the Extend Model

CA Repository for z/OS was designed with a very flexible architecture to allow you to use and customize it to suit your site's particular requirements. All repository-related processes are parameter-driven and externalized from compiled program modules.

Control Tables and Parameter Storage

The repository stores these parameters in DB2 tables called *control* tables. Control tables provide the repository with enough flexibility to support almost any metadata requirement.

CA Repository for z/OS uses different sets of information from the control tables to:

- Build screens
- Display help messages
- Generate menu bars
- Validate user input

Since the repository always refers to these tables during the course of a single edit session, the tables are denormalized to improve performance. De-normalization of the control tables enables the repository to quickly respond to all of your requests.

Extend Model

The repository includes the *Extend* model which is a special model to help you customize the repository. The Extend model consists of special repository object types that enable you to define and customize other repository objects, such as commands, dialogs, screens and so on. As with other repository models, you can use the CA Repository for z/OS editor to edit object instances in the Extend model.

Since the Extend model is used to edit and maintain components that drive the CA Repository for z/OS, you can customize the tools by:

- Using the editor to define and describe repository components
- Using the Import and Build functions from within the editor to move data to and from the control tables
- Using the Quick Map Facility to customize the product

  For several types of changes, Quick Map is the fastest available method for implementing customization.

Several chapters in this guide explain how to customize and maintain metadata models. They also explain how to enroll new users, change help messages, and add new entities.

Note: All activities involving repository metadata models should be tightly regulated, since any changes made to them could adversely affect the way the repository works.

Repository Extension

CA Repository for z/OS provides a fully extensible metadata model and command interface. The repository's Extend facility was designed to enable you to:

- Maintain metadata requirements
- Implement standards
- Support a methodology
- Support CASE implementation
- Provide commands that support user requirements

The Extend Facility should be used in conjunction with the Quick Map Facility. It is easier and faster to do certain types of extensions and customizations in QuickMap rather than using the Build and Import commands from the Extend dialog.

Guidelines for Extending

Any extension should be well thought out. Consider reviewing your customization plan with a CA Repository for z/OS implementation specialist. Make sure that you can service and maintain any extension.

Types of Extensions

The repository's Extend Facility supports several types of extensions. A description of the extension and where it should be performed is shown in the following table.

Action	Function	Where Performed
Add Entities	Defines new entities An example is the entity JOB that could define all batch jobs used within an organization.	Quick Map
Add Relationships	Defines new relationships A relationship is a bidirectional association between two entities that includes context data and has a unique name. A relationship has the full support of the entity editor, but its existence in the repository depends on the existence of its two parent entities: called the **source** and **target**.	Quick Map
Add Associations	Adds new associations.	Quick Map

Action	Function	Where Performed
	This extension is the same as relationships but contains no context data	
Add Dialogs	Defines new groups of entity, relationship, and association types	Quick Map
Add Commands	Develops custom exits from the repository	Extend Dialog
Add Customized Help Messages	Defines or modifies error and help messages	Extend Dialog
Add Code Table Validation and Help	Provides validation for values entered as entity attributes and provides users with lists of valid values	Extend Dialog
Add Domains to Entity and Relationship Types	Adds new domains to entity and relationship types Domains simplify entity definitions by allowing an organization to group attribute values that are frequently used together.	Extend Dialog

Metadata Security

This section discusses the implementation options for metadata security in the repository.

Types of Security

The repository provides significant modularity to meet an organization's metadata security requirements. The following table lists the types of security available.

All of these levels of control can be used together to provide a sophisticated system for controlling access to repository data.

Security Control	Limits
Dialog	Access to all entity, relationship, and association types within a dialog.
Entity	Actions that are performed against specific entity, relationship, and association types, regardless of dialog.
Map	Access to attributes types (field level control), by specifying a different map to use in a dialog to which a user has access.
Status	Access to all instances (entities, relationships, or associations) within a specific phase of an application life cycle.
Command	The commands you can issue.

Security Guidelines

Be aware of the following as you implement security:

- Manage security by privilege groups rather than individuals

  This reduces the effort required to add new users
- Be sure that repository administrators have access to all dialogs so that they are able to govern the access allowed to each dialog
- Be sure that primary attributes appear on initial screens

  These attributes should appear on the sections of the screen first displayed.

Populate the Repository with Metadata

The following table lists the methods that you can use to add metadata to the repository:

Method	Uses
Online Entry	The repository CUA compliant dialog. This method is very effective for creating new systems or documentation.
Batch Entry	Existing metadata population methods including: ■ COBOL, PL/I, and Assembler Copybook Scanners ■ IMS PSB and DBD Scanners ■ RDBMS Catalog Synchronization Facilities ■ IBM Data Dictionary Scanner ■ Data Warehousing Tool Scanners ■ IDMS Scanner ■ Program Scanners ■ Job and Procedure Scanners ■ CSECT Scanner ■ Generic Batch Load Facility
CASE Tool Entry	Custom developed interfaces to migrate data from PC CASE tools to the repository.

Maintain Repository Metadata

You must consider the maintenance aspects of metadata *before* extending the base repository. If you intend to store metadata in the repository, automating the process of maintaining metadata will save you time.

For example, to maintain a list of users kept up in a RACF table that is updated by the Security Administrator, use the Batch Load Facility instead of keying in changes to this list of users. The Batch Load Facility helps to ensure the metadata is accurate and up-to-date.

Logical and Physical Object Definition Support

This section describes how to support Logical and Physical Components.

Logical Components Support

The repository manager-compliant meta-metadata model (how data is physically stored in the repository) is used to support your logical model components.

To support logical model components

- Standardize your way of supporting logical model components, such as implementing a single metadata model in the repository
- If you are unable to implement a single logical model because you use multiple products, define a hybrid metadata model that supports all of your logical requirements

Physical Components Support

The base repository provides complete support for DB2 object definition, PL/I, COBOL, and Assembler copybook definition.

To support physical components

- Install the interface for the product that you use, such as ORACLE, Sybase, Microsoft SQL Server, or Teradata RDBMS in addition to DB2
- Extend the base product if you use alternate DBMSs to support these environments

Name Objects and Supporting Standards

The repository provides many options for naming and supporting multiple names of a single object. The following sections briefly discuss the issues of naming and naming standards.

Create Multiple Names for a Single Object

Use the Extend Facility to create multiple unique and enforced names for a single object. For example, in the repository, an ELEMENT instance has a 32-character element name that serves as its key. It also has a 70-character business name that can be used as a descriptor for the element name or as an alternative COBOL, PL/I, or IMS name. Each ELEMENT also has an 30-character DB2 name and an 8-character Assembler name. Each of these names can be used as the default field name in relationships

About Aliases

The following table lists the types of aliases:

Alias	Definition
True Alias	An entity that (with the exception of its name) has the same attributes as another entity. A true alias is created by defining an ALIAS entity and linking it to another ELEMENT, GROUP, or ALIAS entity through a PHY AL association. An element, alias, or group can be linked to many aliases. During generation, the name of the alias is used with the attributes of the physical element.
Logical Alias	An entity that supports the same business name as another entity, but has a different name and attributes. For example, *CUSTNUM* (8 CHAR) and *CUSTOMER-NUMBER* (10-CHAR) are logical aliases of each other.

Entity Versions

A *version* of an entity is defined as an entity that has the same name and life cycle status of another entity, but has different attributes. The repository provides for +32767 versions of an entity within a single life-cycle status.

Prefixes

You can define standard prefixes that are varied at copybook generation time. These prefixes are implemented through the repository's copybook support, which includes RECORDS, and IMS SEGMENTS.

Dashes, Underscores and Spaces

You can chose whether you want to use dashes, underscores or spaces as the naming delimiter. The repository substitutes the appropriate delimiter at generation time.

Name Generation and Naming Standards

The repository provides a way to integrate an organization-developed naming exit to define object names. This exit is executed on object insert or update and ensures that only valid abbreviations are used. These abbreviations are stored and maintained in the base repository.

Note that the repository provides several ways for you to communicate and enforce naming standards. For more information, see Entity and Attribute Name Generation in the chapter "Using the Relational Translator."

Repository Maintenance

The CA Repository for z/OS requires periodic maintenance and tuning in order to perform efficiently.

Note: This guide addresses DB2 performance issues only for CA Repository for z/OS.

You can use simple DB2 maintenance procedures such as REORGs, RUNSTATs, and REBINDs for most repository performance issues. However, some TSO tuning may be required if an exceptionally high number of users are accessing the repository simultaneously.

The repository also requires a back-up procedure to ensure the smooth operation of tools in the event of a system failure. Standard back-up procedures for DASD and DB2 tables are sufficient. Since the repository may contain vital information about your organization, you should perform manual or job-scheduled backups on a regular basis.

Note: For the DB2 aspects of the repository, including specific procedures regarding backup, recovery, performance tuning, and security, see the chapter "Backup, Recovery, and Performance Tuning" in this guide.

Chapter 2: The Extend Model

This chapter explains the Extend model objects and dialogs that you need to know to understand the features of the Extend model.

This section contains the following topics:

The Extend Model (see page 25)
Extend Model Object Types (see page 25)
The Extend Dialogs (see page 32)
Extend Commands (see page 33)

The Extend Model

The CA Repository uses Entity-Relationship Models (E-R models) to support its data storage and tool requirements.

Each repository tool uses a unique subset of the repository's data model consisting of product-specific object types. The repository even uses an Entity-Relationship model to support customization of the repository itself. This particular model, the Extend Model, consists of a set of entity, relationship, and association types that are used to define the meta models, including object types, privileges, users, messages as well as menus and commands.

As the Repository Administrator, you can manipulate and maintain Extend model entities just like any other repository entities. Once defined, you can move Extend model instances to and from the CA Repository control tables with a set of special commands provided for that purpose. The movement of these entities to and from these control tables directly affects the operation of repository tools.

Extend Model Object Types

Each object type represents a group of characteristics that define the entities, attributes, commands, and windows available to users of the repository. Customizing the repository consists of:

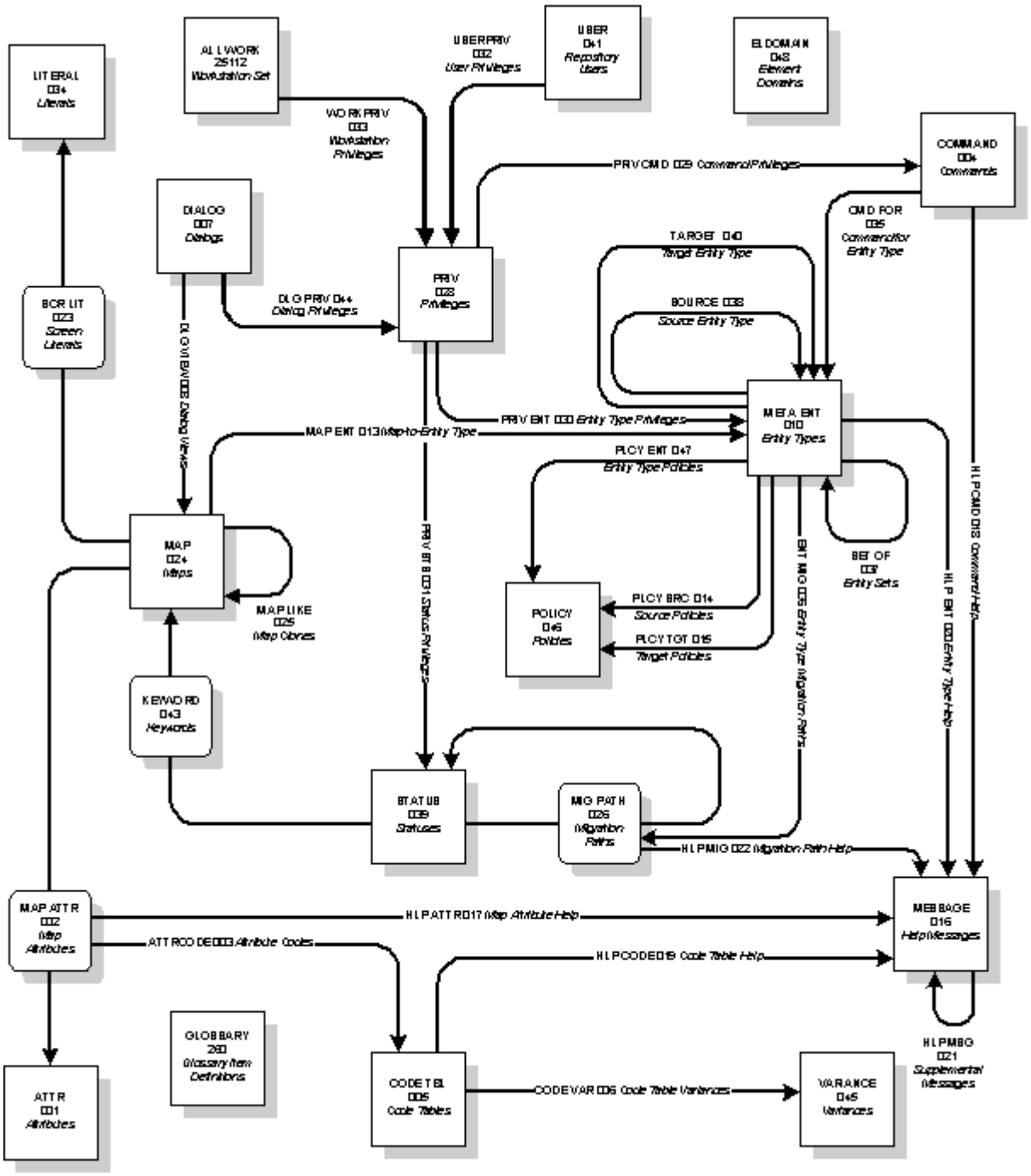
1. Defining one or more instances within one or more of these object types.
2. Moving these definitions to the repository control tables.

CA Repository permits you to customize the repository to fit your site-specific requirements by allowing you to change Extend model instances. You can add:

- Functions to the menu bar
- Commands to pull-down menus
- You can even create entirely new entity types

The three major components within the repository (that is, entity, relationship, and association types) are depicted in the Extend model in different ways. The exact nature of each of these components, and the differences between them, is discussed in detail in the *CA Repository for z/OS Product Guide.* However, the existence of both source and target entities is required prior to the creation of either relationships or associations.

The following diagram shows the repository entity, relationship, and association types that make up the Extend model.



Each of the Extend components is described in the following table. More detailed descriptions are provided in the *CA Repository for z/OS Product Guide*.

Object	Function
ATTR	Defines the attribute types of the various entity, relationship, and association types in the repository Each attribute defined in the Extend model represents the physical characteristics used to store the attribute data within DB2. Most of the attributes defined are used as DB2 columns during the physical implementation of the metadata model.
ATTRCODE	Joins an existing MAP ATTR relationship (the *usage* of a particular attribute) to an existing CODE TBL entity (a set of valid values) In addition to providing validation of data entered into attribute fields, these associations provide selection lists of valid values from the appropriate code table when question marks are entered into attribute input fields of the Edit window.
CMD FOR	Links a COMMAND (command) to one or more META ENT (entity types) These associations determine which entity types a command can act upon. A command that is not linked to an entity type through this association is *entity independent* (it can be used against all entity types).
CODE TBL	Defines repository code tables Code tables are used by the repository to define a set of possible values for an attribute type. Code tables are used by the I/O module to perform validation of attribute values and by the repository to provide attribute value selection lists.
CODE VAR	Joins a CODE TBL entity (code table) to one or more VARIANCE entities (valid values) The CODE TBL entities define the characteristics of the code tables themselves, and the VARIANCE entities contain the actual values that populate each code table. By linking the two types, CODE VAR associations determine which values belong to which code table.
COMMAND	Defines all repository user-specified directives or commands These commands represent both the instructions internal to the repository and those that are added by a user. Corresponding menu bar items and pull-down menu selections are also created as part of the command definition process.
DIALOG	Groups components into logical sets relating to functional or business applications Dialogs are used as the highest level of control over access to repository data.
DLG PRIV	Joins a PRIV entity (privilege definition) to one or more DIALOG entities The repository uses this link to determine which dialogs are accessible to a user when the user is assigned a particular privilege.
DLG VIEW	Connects DIALOG entities to MAPS (physical entity type data) Once established, these associations not only determine which entity types are available when a dialog is selected, but how the Edit screen appears to the user for those entity types.

Object	Function
ENT MIG	Determines which statuses in a migration path are valid for instances of a particular entity type This helps to ensure that instances progress appropriately through an application life cycle.
HLP ATTR	Provides help messages for the attribute types of the various repository entity types These messages appear when a user enters a question mark into an attribute field and selects the help command from the menu bar.
HLP CMD	Provides help messages for each of the repository commands These messages are generated when users type a question mark over the name of a command displayed on a pull-down menu.
HLP CODE	Provides help messages describing the code tables defined in the repository Code table messages are generated when the help command is selected from the menu bar of a Code Table value selection list.
HLP ENT	Provides help messages for the various repository components The Entity messages appear when the help option is selected from the menu bar of the Edit window.
HLP MIG	Provides a help message for a particular migration path
HLP MSG	Provides additional help or further clarification of selected topics Secondary Help messages can be generated by selecting the help command from the menu bar of a Message window.
KEYWORD	Connects a STATUS entity (a *repository usage*) with a MAP (physical entity type) The resulting relationship is assigned a *keyword* that has special meaning to one or more repository tools.
LITERAL	Defines display literals for the vertical Edit windows of the repository These literals can be used to segment screens, as field prompts, or for informative text. Each literal can be used in several different windows, in different locations within those windows, and with different types of highlighting.
MAP	Defines the physical data structure required to store and retrieve the entity information Maps provide entity templates, or views, that describe the information stored and retrieved from the repository. Each map identifies the respective DB2 table used to store entity attributes, extended text, and domain sets. Maps identify the high-level physical storage requirements for an entity. Each MAP entity must be tied to one or more attributes through the MAP ATTR relationship or to another map through the MAP LIKE association before it can be used by an entity type for data storage or retrieval.
MAP ATTR	Links a MAP entity to one or more ATTR entities to define which attribute types make-up a particular map These relationships also contain information concerning the usage of the various

Object	Function
	attribute types by the map to which they are joined. Each map can be linked to up to 300 attribute types through this relationship.
MAP ENT (logical entity type)	Links a MAP to a META ENT. An entity type can be linked to several maps through this association, but a map can only be linked to a single entity.
MAP LIKE	An association type linking one MAP entity to another MAP entity, allowing one map to use all the attributes types (MAP ATTR relationships) of another map Maps joined through this association are identical in terms of attributes. However, the fact that they are each distinct MAP instances allows separate storage tables to be used. The MAP LIKE association overrides any MAP ATTR or SCR LIT relationships that are also linked to the map.
MESSAGE	Defines the repository help and error messages Each message definition contains information about the type of message and the window display parameters. The actual message text is stored as *extended text* for the corresponding instances.
META ENT	An entity type in which each entity type, relationship type, association type, and entity set within the repository is defined as an occurrence Though the actual physical aspects of each component are not a part of the entity definition process, the maintenance policies and rules of cardinality are. The attributes of an entity type are assigned through I/O maps. Because each entity type can have multiple maps, its actual attributes are a combination of all the attributes contained on all the maps linked to that entity type.
MIG PATH	A relationship type that identifies the status paths that can be used when changing an entity from one status to another Entities cannot be updated to a status that is not properly related to their current status through this relationship, unless no migration paths exist at all, in which case, no status checking is done.
PLCY ENT	Links POLICY instances to META ENT instances (entity types); identifies a policy to use when entities of that particular entity type are inserted, updated, or deleted
PLCY SRC	Joins a POLICY to a META ENT association or relationship type that identifies a policy to use for source entities of a particular association or relationship type during insert and update processing
PLCY TGT	Joins a POLICY to a META ENT association or relationship type that identifies a policy to use for target entities of a particular association or relationship type during insert, update, and delete processing
POLICY	Defines SQL-like statements that enforce certain constraints on occurrences of a particular entity type during I/O processing Through three separate association types, policies can be linked to entity types to control the occurrences of any entity type, its source, or its target.

Object	Function
PRIV	Defines privileges that can be subsequently assigned to repository users Through a series of DLG PRIV, PRIV CMD, PRIV ENT, PRIV STS and WORKPRIV associations, each PRIV entity represents a set of commands that can be used against occurrences of certain entity types with particular statuses. Associating a user to a PRIV effectively allows that user to access any commands associated with that privilege. The PRIV entity type is a central access control point for the repository.
PRIV CMD	Joins a PRIV entity (privilege definition) to one or more COMMAND entities The repository uses this link to determine which commands to make accessible to a user that is assigned a particular privilege. PRIV CMD associations are *essential*; a privilege authorizes use *only* of those commands that it has linked through PRIV CMD associations. Each command linked to a privilege is valid for all entities and statuses, unless the privilege was limited by PRIV ENT and PRIV STS associations.
PRIV ENT	Links a PRIV (privilege) with an META ENT entity type The repository uses this link to establish to which entity types a user has access when assigned to a particular privilege. Each entity type linked to that privilege is valid for all commands and statuses linked to that privilege. Unlike PRIV CMD associations, PRIV ENT associations are *restrictive* rather than essential. They are used when a privilege is limited to a specific number of entity types. If no PRIV ENT associations are linked to a privilege, that privilege is valid for all entity types.
PRIV STS	Links a PRIV entity (privilege) to a STATUS entity The repository uses this link to establish the statuses to which a user has access when assigned a particular privilege. Each status linked to that privilege is valid for all commands and entity types linked to that privilege. Like PRIV ENT associations, PRIV STS associations are *restrictive*: they are used when a privilege is limited to specific statuses. If no PRIV STS associations are linked to a privilege, that privilege is valid for all statuses.
SCR LIT	Links the MAP (physical data structure) with the LITERAL (screen display field) Each resultant screen literal contains information about the position of the literal on the vertical Editor window and its highlight characteristics. The horizontal screen literal is also specified as part of these relationships.
SET OF	Links an entity type that was defined as a set with the entity types that are to comprise that set Entity sets can be composed of both entity types and relationship types but not association types.
SOURCE	Links a META ENT (association or relationship type) to a META ENT (entity or relationship type) to identify the source entity type of the association or relationship type

Object	Function
STATUS	Defines statuses that can represent a project, stage, or phase of an application life cycle The repository uses statuses to control user access, entity definition, and entity migration.
TARGET	Links a META ENT (association or relationship type) to a META ENT (entity or relationship type) to identify the target entity type of the association or relationship type
USER	Defines users to the system As part of the USER definition process, each person that is to use the repository must be defined as a separate and unique USER instance. The repository uses these USER instances to provide access to the various tool functions.
USERPRIV	Links USER entities (users) to PRIV entities (privileges). These associations determine which users have access to which privileges (sets of commands).
VARIANCE	Defines one possible variation (value) that you can assign to an attribute Variances are linked to CODE TBL (code tables) to provide selection lists and validation for a particular attribute type. Each variance can be represented in several different ways to allow for differences in the types of code tables that may use that particular variance.
ALL WORK	Contains WORKSTN, WORKPROJ, WORKSUBJ, WORKPACK, STEWARD, DOCUMENT, and CATEGORY entity types These are used to maintain workstations or collections of instances within the repository.

The Extend Dialogs

A CA Repository *dialog* is a group of entity, relationship, or association types that together fit a functional or business requirement. All the entity, relationship, and association types found in the Extend model are grouped together in the Extend dialog so you can maintain the repository using this dialog.

Due to the number of entity, relationship, and association types in the Extend dialog, several sub-dialogs are provided, each of which contains the necessary object types to maintain or customize a particular functional aspect of the repository.

The following table describes these sub-models.

Sub-Model	Usage
COMMAND	Adds new menus and commands
DIALOG	Adds new dialogs
ENTITY	Adds and customizing repository object types
HELPMSG	Adds and customizing CA Repository help and error messages
MAP	Defines new repository Maps
PRIV	Defines privileges for access to repository object types, dialogs, statuses, and commands
STATUS	Defines repository statuses
USERS	Defines repository users

Extend Commands

Extend commands are a group of commands that help a Repository Administrator customize and maintain the CA Repository.

Along with other features, these include commands allow you to:

- Move data between the Extend model and the repository control tables
- Preview repository messages
- Generate DDL for creating DB2 storage tables

To access the Extend commands

1. From the CA Repository CLIST, ARZOS, select the VIEW, DIALOG command to access the Extend dialog.
2. Choose VIEW, TYPE to access one of the following object types:

   - CODE TBL
   - COMMAND
   - DIALOG
   - DLG VIEW
   - KEYWORD
   - MAP
   - MESSAGE
   - META ENT
   - MIG PATH
   - PRIV
   - STATUS
   - USER
   - ALL WORK

   Each of these entity types is considered the *primary entity type* for a group of object types that support a particular functional aspect of the repository. Processing one of the previous entity types using the Extend commands adds or removes data as required from any subordinate entity types.

   The graphic in the Understanding Extend Model Object Types earlier in this chapter uses color to show which object types belong to a particular functional group (the primary entity types appear as solid-colored objects).

1. These are the available Extend commands; each command is discussed in detail in the sections that follow.

   - BUILD
   - IMPORT
   - REMOVE
   - VIEWMSG
   - CREATE
   - RECREATE
   - NEXTID
   - GRANT
   - REVOKE
   - COMPSTAT
   - STATMAP
   -

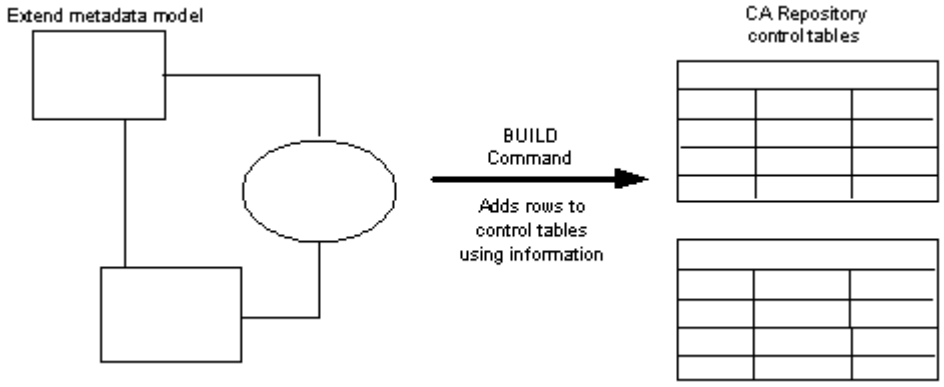
Although you use the Extend model to define repository objects, it is the movement of these definitions to the control tables that actually implements your new commands, privileges, object types, and so on.

The BUILD Command

The BUILD command (accessed using SYSTEM, EXTEND, BUILD), populates the repository control tables using object definitions stored in the Extend model.

The BUILD command:

- Automatically extracts necessary data from all relevant objects
- Inserts rows into the control tables as required



You can execute the BUILD command against any of the following entity types.

- CODE TBL
- COMMAND
- DIALOG
- DLG VIEW
- KEYWORD
- MAP
- MESSAGE
- META ENT
- MIG PATH
- PRIV
- STATUS
- USER
- ALL WORK

The BUILD command takes data from the objects in the extend model and updates the control tables. The objects processed in the Build command (sub-models), are listed in their respective sections of this guide. Some components of a particular sub-model may require components that are part of a different sub-model. Privileges, for example, are linked using associations to commands, entity types, statuses, dialogs, and workstations-all of which belong to separate sub-models. You cannot successfully build a privilege unless you add the COMMAND, META ENT, STATUS, DIALOG, and WORKSTATION (entities linked to that privilege) to the control tables. When building repository components, add those entities used by other entities before building the entities that require them.

Note: There is no Extend table for workstations. Therefore, it is not necessary to run a workstation BUILD prior to a privilege BUILD. When you execute the BUILD command, the CA Repository displays a message describing the processing:

```
SIZE-----------------    MODULE: DBXDISP   -------------------MAX
|                                                             |
|  MESSAGE: EXT00003                                          |
|                                                             |
|  THE AR/ZOS EXTEND FUNCTION IS MOVING DATA FROM THE EXTEND  |
|  META MODEL TO THE CONTROL TABLES. THE BUILD COMMAND IS     |
|  CURRENTLY PROCESSING.                                      |
|                                                             |
|  ENTITY TYPE: <UNKNOWN>, NAME:<UNKNOWN>, STATUS: <UNKNOWN>  |
|  VERSION: <UNKNOWN>                                         |
|                                                             |
|  ******************END OF MESSAGE**************************  |
---------------------------------------------------------------
```

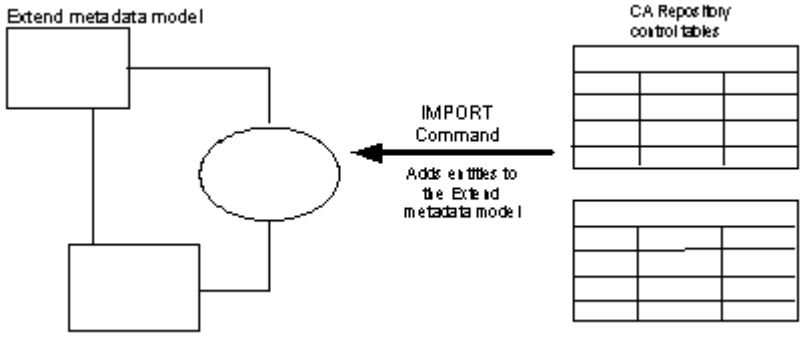
If the build is successful, a complete message appears as shown below.

```
SIZE-----------------   BUILD RESULTS   ----------------------MAX
|  HELP                                                       |
|  GED1,DBXT,00**** COMPLETE ****                             |
---------------------------------------------------------------
```

Note: If you have previously added an entity to the control tables, subsequent building of that entity overwrite earlier information. When using BUILD, take precautionary measures to prevent unexpected loss of control table definitions.

The IMPORT Command

The IMPORT command (accessed using SYSTEM.EXTEND.IMPORT), is useful when you want to modify a repository component that is not defined in the Extend model. This command uses data from control tables to create instances in the EXTEND dialog.



You can use the IMPORT command against any of the Extend model object types in the list that follows. When you import from one of these object types, IMPORT automatically creates:

- Any subordinate entities, relationships, and associations that are within the current sub-model
- Any associations to subordinate entities or relationships that are not a part of the current sub-model if those entities or relationships already exist in the EXTEND dialog data tables

CODE TBL	COMMAND	DIALOG
DLG VIEW	KEYWORD	MAP
MESSAGE	META ENT	MIG PATH
PRIV	STATUS	USER
ALL WORK		

Import does not require that you select a particular instance for processing. The IMPORT command displays a list of all control table definitions for the current entity type. You simply have to select the instances to import from this list.

```
SIZE ---------------- AR/ZOS IMPORT ENTITY LIST                    MAX
| CRITERIA PROFILE NAVIGATE HELP                                    *
| SEL   ENT_NAME ENT_T DESCRIPTION                                  |
| ---   -------- ----- ---------------------------------------------|
|  _    AL FOR   101   (TB ALIAS/LOCATION) LINK A TB ALIAS TO LOCATI  |
|  _    ALIAS    202   ALIAS DEFINITION                             |
|  _    ALL ATTR 26006 ALL ATTRIBUTES SET                           |
|  _    ALL ENT  26005 ALL ENTITIES SET                             |
|  _    ALLOWVAL 25781 ALLOWABLE VALUES IN AN ALLOWED VALUE TABLE   |
|  _    ALLPRCES 26004 ALL PROCESSES SET                            |
|  _    APPL     251   APPLICATION                                  |
|  _    APPLCONV 261   APPLICATION TO CONVERSATION ASSOCIATION      |
|  _    APPLJOB  267   APPLICATION TO JOB ASSOCIATION               |
|  _    ARTICLE  76    DOCUMENT APPEARING IN A PERIODICAL ISSUE     |
|  _    ASYNCALL 512   ASYNCHRONOUSLY CALLS (PROGRAM/PROGRAM)       |
|  _    ASYNPARM 516   ASYNC CALL USES PARAMETER (ASYNCALL/RECORD)  |
|  _    AT CO AT 335   ATTRIBUTE TYPE CONSISTS OF ATTRIBUTE TYPE    |
```

Note: If you execute IMPORT followed by the letter Q (for example, SYSTEM, EXTEND, IMPORT Q), a Criteria window appears before the list of control table definitions. This allows you to narrow the list using search criteria.

When using IMPORT, you need to specify a status for the entities it will create. The repository prompts you for a status:

```
SIZE ---------------- IMPORT STATUS                    MAX
| NAVIGATE HELP                                        |
|                                                      |
| ENTER THE STATUS TO IMPORT INTO: TEST----------------|
|                                                      |
```

Depending on the sub-models involved, the source or target entities of a particular relationship or association are not automatically imported from the control tables. For example, when you import a PRIV entity, PRIV CMD associations are only created if the appropriate COMMAND entities already exist in the EXTEND dialog. If all the required entities are not already in the EXTEND dialog, the import results are listed as INCOMPLETE.

The REMOVE Command

The REMOVE command (accessed using SYSTEM, EXTEND, REMOVE), allows you to *deactivate* a particular repository component by removing it from the control tables. Once removed from the control tables, a repository command, privilege, and so on becomes inoperable. The REMOVE command does not remove or otherwise affect any data in the Extend model data tables.

You can execute REMOVE against of any of the following entity types. The CA Repository automatically removes any subordinate objects from the control tables.

CODE TBL	COMMAND	DIALOG
DLG VIEW	KEYWORD	MAP
MESSAGE	META ENT	MIG PATH
PRIV	STATUS	USER
ALL WORK		

You do not have to select a specific entity when using REMOVE. The CA Repository displays a list of all control table definitions for the current entity type-you simply need to select the appropriate definition from this list.

```
SIZE ---------------- AR/ZOS REMOVE ENTITY LIST                    MAX
| CRITERIA PROFILE NAVIGATE HELP                                    *
| SEL   ENT_NAME ENT_T DESCRIPTION                                  |
| ---   -------- ----- ---------------------------------------------|
|  _    AL FOR   101   (TB ALIAS/LOCATION) LINK A TB ALIAS TO LOCATI  |
|  _    ALIAS    202   ALIAS DEFINITION                             |
|  _    ALL ATTR 26006 ALL ATTRIBUTES SET                           |
|  _    ALL ENT  26005 ALL ENTITIES SET                             |
|  _    ALLOWVAL 25781 ALLOWABLE VALUES IN AN ALLOWED VALUE TABLE   |
|  _    ALLPRCES 26004 ALL PROCESSES SET                            |
|  _    APPL     251   APPLICATION                                  |
|  _    APPLCONV 261   APPLICATION TO CONVERSATION ASSOCIATION      |
|  _    APPLJOB  267   APPLICATION TO JOB ASSOCIATION               |
|  _    ARTICLE  76    DOCUMENT APPEARING IN A PERIODICAL ISSUE     |
|  _    ASYNCALL 512   ASYNCHRONOUSLY CALLS (PROGRAM/PROGRAM)       |
|  _    ASYNPARM 516   ASYNC CALL USES PARAMETER (ASYNCALL/RECORD)  |
|  _    AT CO AT 335   ATTRIBUTE TYPE CONSISTS OF ATTRIBUTE TYPE    |
```

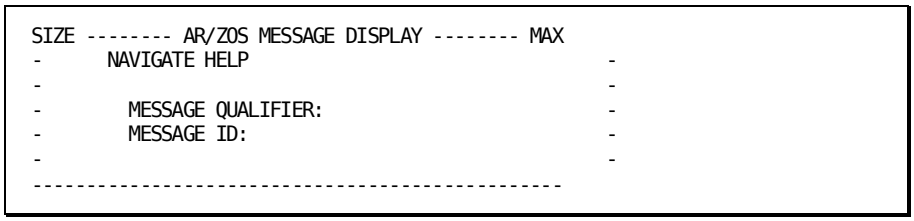
Note: If the number of definitions in the repository is very large, enter the command from the command line using the Q switch (SYSTEM.EXTEND.REMOVE Q) to view a Criteria Selection window before the list of control table definitions. You can then narrow the list of definitions using search criteria.

The VIEWMSG Command

The VIEWMSG command (accessed using SYSTEM, EXTEND, VIEWMSG), is useful when you're adding or customizing the CA Repository error and help messages. VIEWMSG enables you to preview messages you are creating or altering.

To execute the VIEWMSG command

1. From the CA Repository clist, ARZOS, select VIEW, DIALOG, EXTEND. Then select VIEW, TYPE, MESSAGE.
2. Choose SYSTEM.EXTEND.VIEWMSG to display the CA Repository Message Display panel.



3. Enter the appropriate message Qualifier and ID for the message you want to view, then press Enter.

   The CA Repository displays the message according to its default characteristics.

The CREATE Command

The CREATE command(accessed using SYSTEM, EXTEND, CREATE), allows you to create the physical DB2 data structures required to store instances of a repository object type. This command is useful when you are adding new object types to your repository models.

Before you execute the CREATE command you must:

1. Define and build an entity type and its respective maps.
2. Link the entity type and maps to a dialog.
3. Build or rebuild the dialog.

Note: The CREATE command will not work properly until you complete these steps.

To execute the CREATE command

1. From the CA Repository clist, ARZOS, select VIEW, DIALOG, EXTEND. Then select VIEW, TYPE, MESSAGE.
2. Select the instance defining the entity or relationship type for which you need to create a storage table.
3. Execute the SYSTEM.EXTEND.CREATE command.

   An Edit screen containing the CREATE TABLE syntax for the entity type you selected appears.

   Note: The CREATE command is also available from the Quick Map Facility.

   Edit the JCL to include the proper database and storage group names (site-specific changes to the JCL may also be required).
4. Enter SUB on the command line.

   This submits the job.
5. Use the END command (PF3) to exit the edit session.

The RECREATE Command

The RECREATE command (accessed using SYSTEM, EXTEND, RECREATE) is useful when you are modifying a repository object type where you need to ALTER its underlying DB2 storage table. The RECREATE command does the following:

- Unload data from the current table into a temporary file
- Generate and store syntax for reloading the table
- Drop the current table
- Create a new table using the data currently in the repository and grant users access to it
- Reload the data

To execute the RECREATE command

1. In the Extend dialog, choose VIEW, TYPE to access the META ENT object type.
2. Choose VIEW, LIST, ENTITY to select the entity defining the entity/relationship for which you want to rebuild the table.
3. To display an edit screen containing the RECREATE TABLE syntax for the selected object type, choose SYSTEM.EXTEND.RECREATE.

   Note: The RECREATE command is also available from the Quick Map Facility.
4. Edit the JCL prior to submission to include the proper database and storage group names-other site-specific changes to the JCL may also be required.

5. Edit the load syntax if:
   - An existing column was dropped or renamed
   - The length of an existing column was decreased
   - The data type of an existing column was changed

   Note: Before you submit the JCL, back up the data in the original table. If a problem occurs during processing, use the backup to restore the table. The LOAD syntax is stored in the SYSPUNCH data set allocated in the UNLOAD step.
6. Enter SUB on the command line to submit the JCL.
7. Use the END command (press PF3) to exit the Edit screen.
8. It is recommended that you take an image copy of the new table at this point. Previous image copies are no longer usable, since the structure of the table has changed.

The NEXTID Command

Use the NEXTID command, accessed using SYSTEM, EXTEND, NEXTID, to identify which MAP and ENTITY ID's are available when defining a new map or entity type.

You must define new entity types and associated maps when you build custom metadata models tailored to site-specific applications. Entity types and maps require a unique ID that is used internally by the repository during processing. Maps require a MAP ID, while entity types require an ENTITY ID. You must specify both types of IDs during the respective definitions of META ENT and MAP entities.

To execute the NEXTID command

1. Choose VIEW, TYPE to select the MAP or META ENT object type.
2. Execute SYSTEM, EXTEND, NEXTID.

   A message similar to the one below appears.

```
SIZE --------------------MODULE: DBXNEXT-------- MAX
  _
  MESSAGE: EXT0016                                                  -
                                                                    -
  THE NEXT AVAILABLE ENTITY TYPE FOUND IN THE DB EXCEL EXTEND       -
  MODEL IS 1002                                                     -
-                                                                   -
  THE NEXT AVAILABLE ENTITY TYPE FOUND IN THE DB EXCEL              -
  CONTROL TABLES IS 1001                                            -
-                                                                   -
  **********************END OF MESSAGE************************************
```

Since not every entity or map defined in the metadata model is added to the control tables, the next ID available in the control tables may not be the same as that available in the metadata model. The NEXTID message displays both.

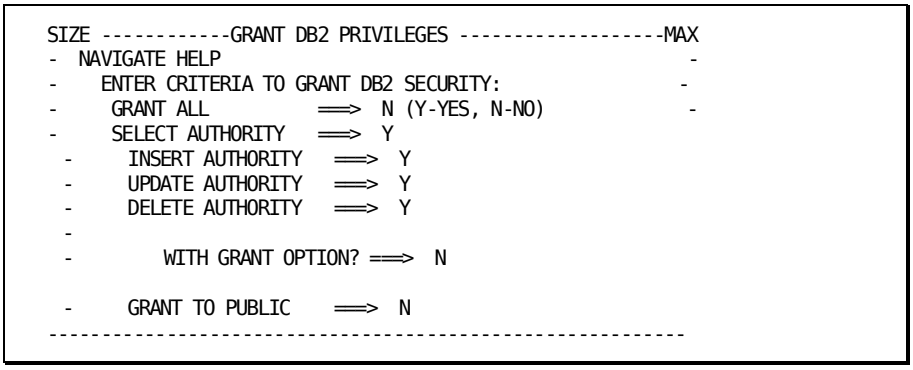
Note: The CA Repository reserves certain entity and map IDs for internal use and future expansion. These include IDs 1-999 and those above 25000. IDs 1000-24999 are not reserved and can be used when making extensions.

The GRANT Command

The SYSTEM.EXTEND.GRANT command enables you to grant DB2 access to users who MUST access entities in the repository. By default, new repository users should have access to all the repository's DB2 tables created with a security specification of GRANT, DELETE, INSERT, SELECT, UPDATE, TO PUBLIC. However, if a particular table does not allows these sort of actions, you can use the GRANT command to provide the appropriate authority.

To use the GRANT command

1. Choose VIEW, TYPE to access the META ENT entity type.
2. Choose SYSTEM, EXTEND, GRANT to open the Grant Privileges screen.



3. Specify the privileges to grant.
4. To view a list of all repository users currently defined in the control tables, press Enter to exit the GRANT privileges screen.
5. Select the users to whom you want to grant privileges to by tagging the appropriate select bytes.

   A list of entity types currently defined to the repository appears. Press Enter.

   Select the desired entity types by tagging the appropriate select byte and press Enter.

Note: Some entity types may use the same storage table. Granting access to any one of them provides access to that table, and thus to all of the entity types using that table.

The REVOKE Command

The REVOKE command (accessed using SYSTEM, EXTEND, REVOKE), allows you to revoke a user's access to repository tables.

To revoke user access to repository tables

1. In the Extend dialog, access the USER entity type.
2. Choose SYSTEM, EXTEND, REVOKE.

   The REVOKE command leads you through a series of screens, each of which prompts you for a different REVOKE parameter. These parameters include:

   - The type of privileges to revoke
   - The users to whom the revoke is to apply
   - The entities (tables) to which access is revoked

Chapter 3: Manage Repository Users

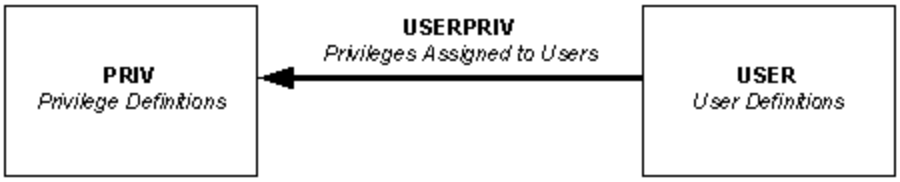
This chapter describes how to use the Extend model to manage repository users.

This section contains the following topics:

The User-Privilege Model-USERPRIV (see page 45)
Add a New User (see page 46)
Modify a User Definition (see page 49)
Extract User Data from the Control Tables (see page 50)
Remove a User (see page 52)

The User-Privilege Model-USERPRIV

The CA Repository uses the objects types shown in the following diagram to define repository users and associate them with one or more privileges.



The following is a list of object types and their purpose.

Object	Purpose
USER	Defines individuals who access the repository. The CA Repository uses USER instances to provide access to repository tools, functions, and metadata.
USERPRIV	Links definitions of repository users (USER instances) to definitions of repository privileges (PRIV instances). These associations determine which users have access to which privileges.
PRIV	Defines repository privileges. The PRIV entity type is not a direct part of managing Repository users. This entity type is discussed in detail in the chapter "Working with Privileges" later in this document.

Add a New User

The following steps required to provide access to the repository to a new user:

1. Insert a USER instance that defines the new user into the Repository.
2. Associate the USER definition with one or more repository privileges.
3. Add the user definition to the repository control tables.

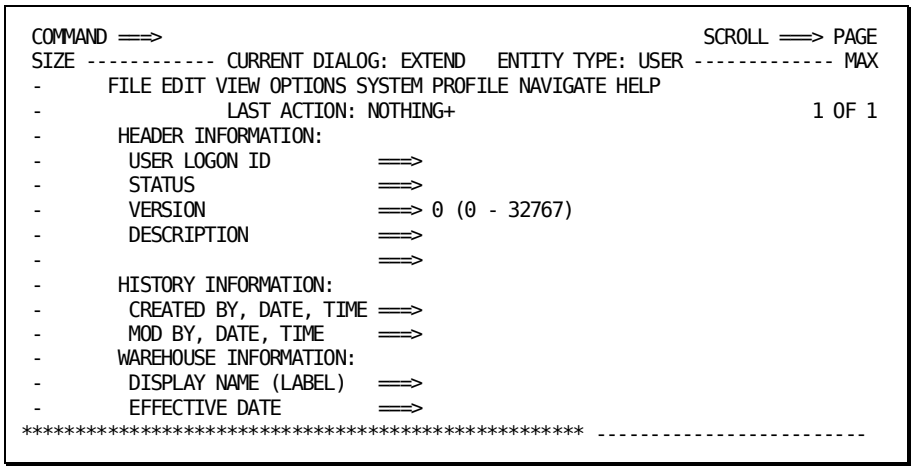
Note: The following sections describe how to perform each of the previous steps.

Insert a USER Instance that Defines a New User into the Repository

The first thing you need to do when adding a new user to the repository is to create an instance of the USER entity type that defines the new user.

To create a new USER instance

1. Select VIEW, DIALOG, EXTEND from the CA Repository CLIST, ARZOS, to access the Extend dialog.
2. Choose VIEW, TYPE, USER to access the object type Edit screen.



3. Enter the appropriate values in each of the USER input fields as specified in the following table. You can ignore any field that is not included.

Field	Enter
User LOGON ID	The TSO ID of the user for which you're defining this instance.
STATUS	A valid repository status for the EXTEND dialog.
VERSION	0 (or the version of your choice).

Field	Enter
NUMBER	
DESCRIPTION	A brief description of the user (for example, the user's name).

4. Choose EDIT, INSERT to insert the USER definition into the repository. Be sure to keep a record of the Logon ID, Status, and Version Number of this instance-you will need this information in the next two procedures.

   Note: You still need to associate this USER instance to one or more privileges *and* add it to the control tables before the associated user is able to access the Repository.

Associate a User with Repository Privileges

After you insert a user definition into the repository, you need to associate that user with a set of repository privileges. Privileges are sets of authorities that provide access to commands, dialogs, and statuses allowing the user to view or perform actions against repository data.

By associating a USER entity with a PRIV entity, you authorize a user access to commands, dialogs, entity types, statuses, and workstations represented by that privilege. For more information about privileges, see the chapter "Working with Privileges" chapter later in this guide.

To associate a user with a set of privileges

1. Choose VIEW, DIALOG, EXTEND to access the Extend dialog.
2. Choose VIEW, TYPE, USERPRIV to access the USERPRIV entity type Edit screen.

```
COMMAND ===>                                                  SCROLL ===> PAGE
SIZE ---------- CURRENT DIALOG: EXTEND   ENTITY TYPE: USERPRIV ----------- MAX
-      FILE EDIT VIEW OPTIONS SYSTEM PROFILE NAVIGATE HELP
-                 LAST ACTION: NOTHING+                              1 OF 1
-       USER INFORMATION:
-        TSO LOGON ID           ===>
-        STATUS                 ===>
-        VERSION NUMBER         ===>    (0 - 32767)
-       PRIVILEGE INFORMATION:
-        PRIVILEGE NAME         ===>
-        STATUS                 ===>
-        VERSION NUMBER         ===>    (0 - 32767)
------------------------------------------------------------------------------
```

3. Enter values into the USERPRIV fields as follows:

Field	Enter
TSO LOGON ID	The TSO LOGON ID of a new user in the Repository section. This is the logon ID you specified in Step 3 in Inserting a USER Instance.
STATUS	The New User's status in the Repository Section. This is the status you specified in Step 3 in Inserting a USER Instance.
VERSION NUMBER	The version you entered in Step 3 in Inserting a USER Instance that Defines a New User in the Repository section.
PRIVILEGE NAME	The name of the PRIV instance that defines the privileges you want to grant to this user.
STATUS	The status of the PRIV instance.
VERSION NUMBER	The version of the PRIV instance.

4. Choose EDIT, INSERT to insert the USERPRIV association in the repository.

   Note: The specified USER and PRIV entities must already exist in the repository before you can insert the USERPRIV association.

To insert another association linking the same USER instance to another PRIV instance. You can associate the USER instance to an unlimited number of privileges.

Add a User Definition to the Control Tables

After you insert a USER instance and associate it to one or more privileges, you need to add the user definition to the control tables. The individual associated with the USER instance will be unable to access the Repository until you add the USER instance to the control tables.

To add a USER definition to the repository control tables

1. Choose EDIT, GOTO to return to the USER Edit screen. The USER instance you used to create USERPRIV associations is automatically selected.
2. Choose SYSTEM, EXTEND, BUILD to add the user to the control tables. The BUILD command adds the user and any associated USERPRIV associations to the control tables. A message indicating the success or failure of the build appears after processing.

   Note: Since certain entity type storage tables may be restricted by DB2 security, a new user may not have access to all the entity types authorized by the assigned privileges. For these situations, you must grant the appropriate authority directly to the new user. For more information, see The GRANT Command in the "The Extend Model" chapter.

   Without further specific authority, a new user should have access to all DB2 tables created with a security specification of GRANT, DELETE, INSERT, SELECT, UPDATE, TO PUBLIC.

Modify a User Definition

As user requirements change, you might need to modify certain aspects of a user definition.

To modify a user definition

1. Enter VIEW.DIALOG to access the Extend dialog in CA Repository.
2. Choose VIEW, TYPE to select the USER entity type.
3. Choose VIEW, LIST, ENTITY to select the USER instance you want to modify.

   Note: If the USER instance is no longer in the Extend model, you can import it from the control tables-see Extract User Data from the Control Tables later in this chapter.
4. If you must change the USER instance itself, make the appropriate changes and choose EDIT, UPDATE to commit the modifications.

5. If you need to change the user's authority:
   a. Choose EDIT, GOTO to select the USERPRIV association type. CA Repository automatically displays all USERPRIV associations that use the USER instance, up to the maximum number of instances that can be displayed in the Editor at one time.
   b. Insert or delete USERPRIV associations as necessary to link the user to the appropriate privileges.
6. Choose EDIT, GOTO. The USER entity type reappears, reselect the USER instance.
7. Choose SYSTEM, EXTEND, BUILD to rebuild the USER entity. This command replaces the user and all its associated objects in the control tables with the modified versions.

Extract User Data from the Control Tables

If the user data in the Extend model no longer exists or was altered, you can use the IMPORT command to bring data directly from the control tables to the repository storage tables. Use the following procedure to import data from control tables to storage tables.

To import data from control tables to storage tables

1. Choose VIEW, DIALOG to select the Extend dialog.
2. Choose VIEW, TYPE to select the USER entity type. The USER Edit screen appears.

   Note: To add or remove any data from the control tables that pertain to users (USER and USERPRIV), always use the USER entity type.

3. Choose SYSTEM, EXTEND, IMPORT.

   A list of user definitions currently in the control tables appears, similar to the following list.

```
SIZE ------------ CURRENT DIALOG: EXTEND   ENTITY TYPE: USER ------------- MAX
-  FILE EDIT VIEW OPTIONS SYSTEM PROFILE NAVIGATE HELP
-            LAST ACTION | DEFAULTS   |
-   HEADER INFORMATION:   | COMMANDS   |
-    SIZE AR/ZOS IMPORT U MAX ITERIA   |
-    | CRITERIA PROFILE NAV * GO       |
-    | SEL   TSO_LOGO       * TEND   > |      0 - 32767)
-    | ---   --------       | -----------------
-    |       _    CICS330A  |    -      =>
-    |       _    PDAAC     |    -
-    |       _    PDAACA    |    -
-    |       _    PDAACB    |    -
-    |       _    PDBMU     |    -
-    |       _    PDFAL     |    -
-    |       _    PDFALA    |    -
-    |       _    PDJHL     |    -
-    |       _    PDJKLA    |    -
```

4. Enter **S** in the SEL field of each user you want to import; then press Enter.

5. Enter a valid status for the EXTEND dialog at the prompt then press Enter.

   Notes:

   - If the selected user is not in the repository model, a progress message appears stating that the import request is being processed. Processing is very rapid, so the progress message may be displayed and removed before you can read it. Once the process message disappears, a message is displayed stating that the import has completed.
   - If the user already exists in the repository model but was modified, the import progresses as explained previously. However, the system overwrites the existing USER entity with a new one

   You can now use the Editor to select and make modifications to the USER instance and its associated USERPRIV instances as needed.

   Note: Make certain all necessary USERPRIV associations exist in the Repository model *before* rebuilding a USER entity**.**

Remove a User

When it is necessary to revoke all access to the repository for a particular user, you need to remove the appropriate USER instance from the repository control tables as described in the following procedure.

To remove a user from the control tables

1. Choose VIEW, DIALOG to select the EXTEND dialog.
2. Use VIEW, TYPE to select the USER entity type. The USER Edit screen appears. For an example of this screen, see the section Insert a USER Instance that Defines a New User in the Repository earlier in this chapter.
3. Use the SYSTEM, EXTEND, REMOVE command to display a list of all user definitions currently in the control tables. This list looks similar to the Import list shown in the previous section.
4. Enter **S** in the SEL field of each user you want to remove; then press Enter. A message appears indicating that a user definition was successfully removed from the control tables. The associated user ID is now unable to access the repository.

   Note: Selecting a user ID from this list of user definitions removes that user definition from the control tables and eliminates any associations of the user to privileges. Any USER and USERPRIV definitions in the Extend model are unaffected.

Chapter 4: Working with Privileges

In order to define and maintain repository privileges, you must be familiar with repository edit procedures and navigation.

Privileges serve the following purposes:

- Control user access to repository data
- Allow access to one or more commands, dialogs, entity types, statuses, or workstations

This chapter explains how to set up and maintain CA Repository privileges. If you are new to repository edit procedures and navigation, see the *CA Repository for z/OS Product Guide*.

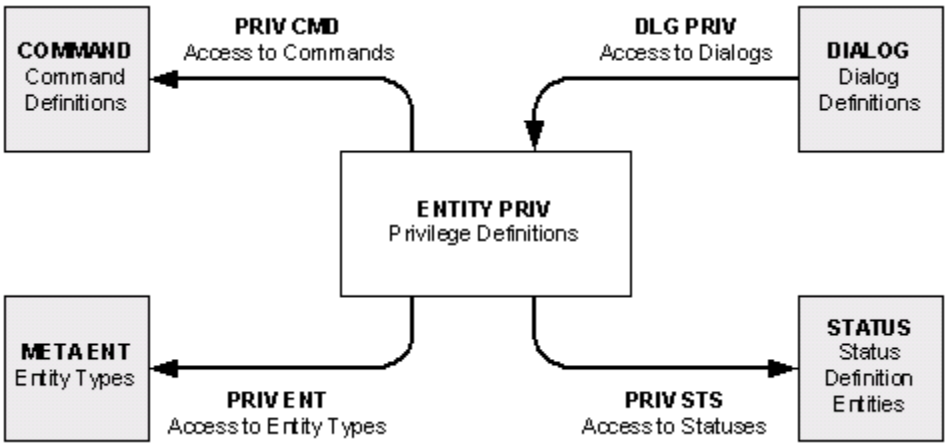
This section contains the following topics:

The Privilege Model (see page 53)
How to Add Privileges (see page 55)
Modify a Privilege (see page 61)
Extract Altered Privilege Data (see page 62)
Remove a Privilege (see page 63)
How to Set Up Default Privileges for New Users (see page 63)
Command Checklist (see page 65)

The Privilege Model

The EXTEND model object types used for administering repository privileges include the PRIV entity type and the DLG PRIV, PRIV CMD, PRIV ENT, PRIV STS and WORK PRIV association types. The shaded components, in the following diagram represent entity types that must exist for privileges to be established.

The following diagram shows the Extend Model Object Types for Defining Repository Privileges



The purpose of each of the Privilege model entity and association types is as follows.

Type	Function
PRIV	Defines privileges and adds privileges to the control tables. A single privilege can be linked to any number of commands, dialogs, entity types, statuses, and workstations.
PRIV CMD	Links privilege definitions to command definitions to determine the commands to which a user has access when assigned a particular privilege.
PRIV ENT	Links a privilege definition to one or more entity type definitions to limit which entity types a user can access when assigned a particular privilege.
PRIV STS	Links a privilege definition to a status definition to determine which statuses a user can access when assigned a particular privilege.
DLG PRIV	Links a dialog to a privilege to determine which dialogs a user can access when assigned a particular privilege.
WORKPRIV	Links a workstation to a privilege to determine which workstations a user can access when assigned a particular privilege.

How to Add Privileges

The following is a list of the steps in the procedure you must use to define and implement repository privileges. Each of the steps is explained in detail in the sections that follow:

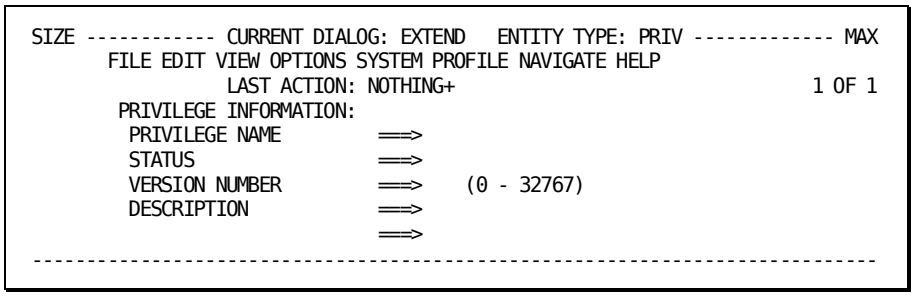
1. Insert (define) a PRIV instance.
2. Link the PRIV instance to the commands that must be accessible through the privilege by inserting (defining) one or more PRIV CMD associations.
3. Link the PRIV instance to any dialogs that must be accessible through the privilege by inserting (defining) one or more DLG PRIV associations.
4. Link the PRIV instance to entity types (if you want to restrict the privilege to certain entity types) by inserting (defining) one or more PRIV ENT associations.
5. Link the PRIV instance to certain statuses (if you want to restrict the privilege to certain statuses) by inserting (defining) one or more PRIV STS associations.
6. Re-access the PRIV entity type and reselect the PRIV instance you created in Step 3.
7. Choose SYSTEM, EXTEND, BUILD to add the privilege and its associations to the control tables.

Insert (Define) a Privilege Instance

The first action you must take when defining a privilege is to insert a PRIV instance. This instance will be a central link for a group of associations tied to command, dialog, entity type, status definitions, and workstation.

To define a privilege instance

1. From within the CA Repository, choose VIEW, DIALOG to access the EXTEND dialog.
2. Choose VIEW, TYPE to access the PRIV Edit panel.



3. Enter values into the PRIV fields as follows:

Field	Enter
PRIVILEGE NAME	A name for the PRIV entity instance.
STATUS	The default status for the extend dialog is EXTP.
VERSION NUMBER	The PRIV entity instance.
DESCRIPTION	A description of the privilege.

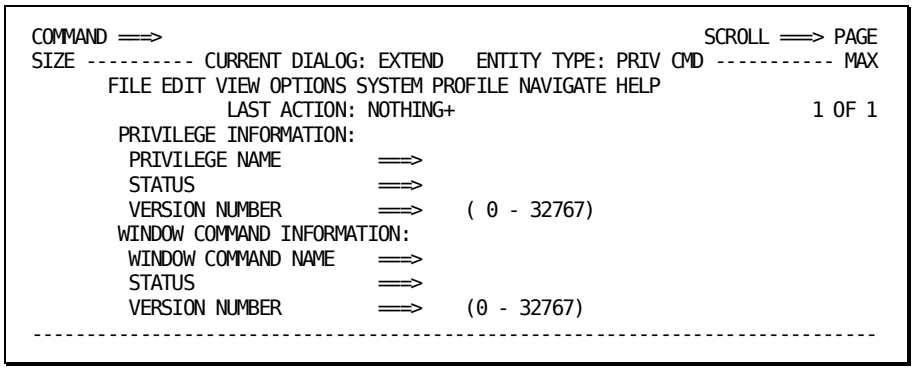
4. Choose EDIT, INSERT to add the PRIV entity to the repository.

Link the Privilege to Commands

After you define a privilege, you must link it to one more commands.

To link privileges to commands

1. Choose VIEW, TYPE. The PRIV CMD Edit screen appears.



2. In the Privilege Information section, enter the name, status, and version of the PRIV instance you just created.
3. In the Window Command Information section, enter the name, status, and version of a COMMAND instance defining a command that you would like to access using this privilege. A checklist of available commands is provided in the Command Checklist, described later in this chapter.

   Note: You can enter information directly into the PRIV CMD input fields or choose VIEW, LIST, SOURCE and VIEW, LIST, TARGET command strings to retrieve the necessary data.
4. Choose EDIT, INSERT to add the PRIV CMD association to the repository.
5. If the new privilege will provide access to more than one command, repeat Steps 2-4 to link the PRIV instance to the appropriate COMMAND definitions.

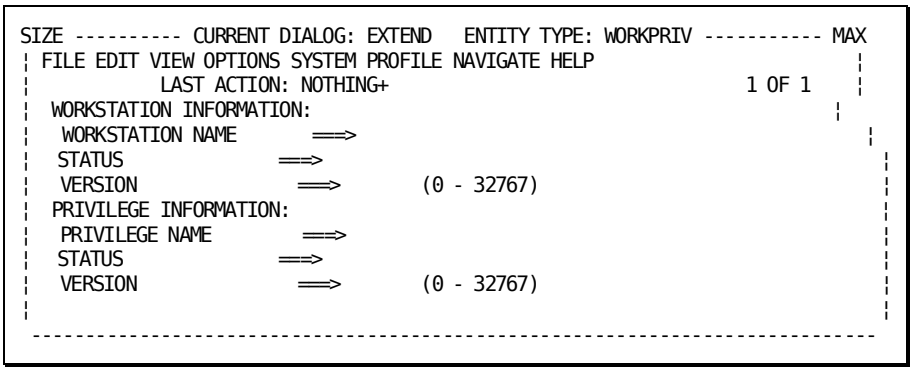
Link the Privilege to Workstations

To determine which CA Repository workstations a privilege can access, you must link the PRIV instance to one or more WORKSTATION instances. You need to define and insert a separate association linking the privilege to each workstation that will be available through the new privilege.

Note: If a Steward type workstation, is not associated with any privilege then the steward can not approve any pending changes to objects associated with the steward workstation in Webstation Option.

To link privileges to Workstation

1. Choose VIEW, TYPE to access the WORKPRIV association type workstations.



2. In the Workstation Information section, enter the name, status, and version number of a WORKSTATION instance defining a workstation that you want to make accessible through this privilege.
3. In the Privilege Information section, enter the name, status, and version of the PRIV you are creating.
4. Choose EDIT, INSERT to add the WORKPRIV association to the Repository. Repeat Steps 2 - 4 for any additional workstations you want to make available through the privilege.

Link the Privilege to Dialogs

To determine which CA Repository dialogs a privilege can access, you must link the PRIV instance to one or more DIALOG instances. You need to define and insert a separate association linking the privilege to each dialog that will be available through the new privilege.

Note: If a privilege is not linked to any dialogs, that privilege does not provide access to any dialogs.

To link privileges to dialogs

1. Choose VIEW, TYPE to access the DLG PRIV association type dialog.

```
COMMAND ===>                                                  SCROLL ===> PAGE
SIZE ---------- CURRENT DIALOG: EXTEND   ENTITY TYPE: DLG PRIV ----------- MAX
       FILE EDIT VIEW OPTIONS SYSTEM PROFILE NAVIGATE HELP
                 LAST ACTION: NOTHING+                                1 OF 1
        DIALOG INFORMATION:
         DIALOG NAME           ===>
         STATUS                ===>
         VERSION NUMBER        ===>    (0 - 32767)
        PRIVILEGE INFORMATION:
         PRIVILEGE NAME        ===>
         STATUS                ===>
         VERSION NUMBER        ===>    (0 - 32767)
------------------------------------------------------------------------------
```

2. In the Dialog Information section, enter the name, status, and version number of a DIALOG instance defining a dialog that you want to make accessible through this privilege.
3. In the Privilege Information section, enter the name, status, and version of the PRIV you are creating.
4. Choose EDIT, INSERT to add the DLG PRIV association to the Repository.

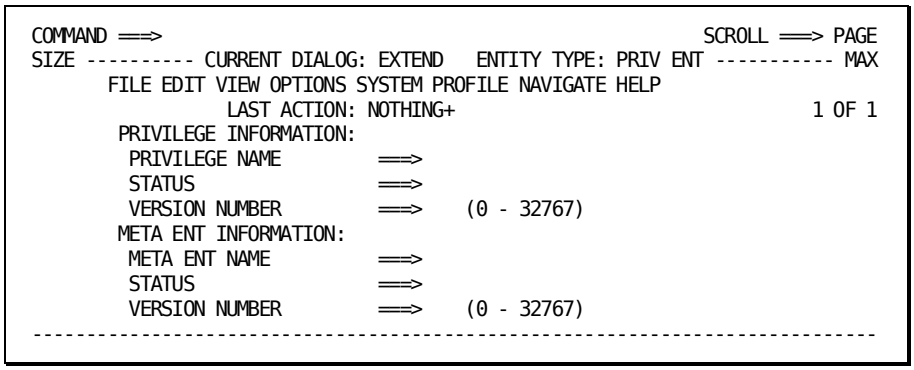
Repeat Steps 2 - 4 for any additional dialogs you want to make available through the privilege.

Limit the Privilege to Certain Entity Types

If you want a privilege to be limited to certain CA Repository object types, you will need to link the PRIV instance to one more META ENT instances via PRIV ENT associations. Linking privileges to entity types is *not* required-you only need to do this if you want to restrict the privilege to specific entity types. If you do not link the PRIV instance to META ENT instances, the privilege provides access to all entity types.

To link privileges to entity types

1. Choose VIEW, TYPE. The PRIV ENT association type dialog appears.



2. In the Privilege Information section, enter the name, status, and version of the PRIV you are creating.
3. In the Meta Ent Information section, enter the name, status, and version of a META ENT instance defining a repository object type that you would like to be accessible through this privilege.
4. Choose EDIT, INSERT to add the PRIV ENT association to the repository.
5. Repeat Steps 2-4 for each object type you want the privilege of being able to access.

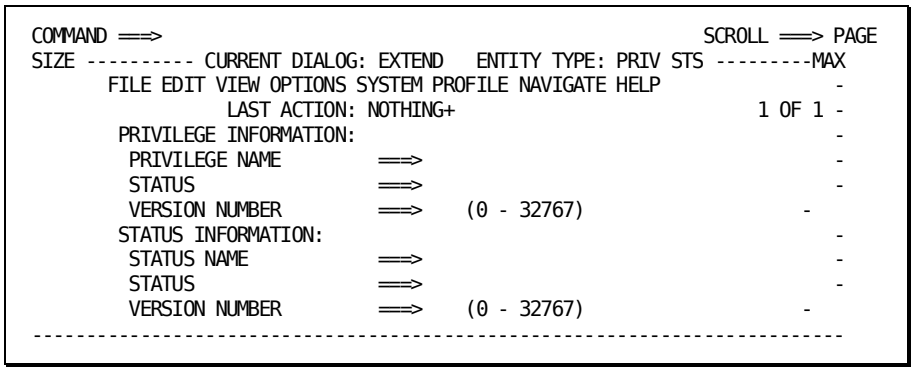
Limit a Privilege to Certain Statuses

If you want to limit a privilege to certain CA Repository statuses, you must link the PRIV instance to one more STATUS instances using PRIV STS associations.

Note: Linking privileges to statuses is not required-you only do this if you want to restrict the privilege to instances having a particular status. If you do not link the PRIV instance to STATUS instances, the privilege provides access to all repository statuses.

To link privileges to statuses

1. In the EXTEND dialog, choose VIEW, TYPE. The PRIV STS association type dialog appears.



2. In the Privilege Information section, enter the name, status, and version of the PRIV you are creating.
3. In the Status Information section, enter the name, status, and version of a STATUS instance defining a repository status that you want to be accessible through this privilege.
4. Choose EDIT, INSERT to add the PRIV STS association to the repository.

Repeat Steps 2-4 for each status you want to be accessible when using the privilege.

Add Privileges to the Control Tables

After you define a privilege and linked it to commands, dialogs, entity types, statuses, and workstations, you need to add the privilege to the repository's control tables. The privilege will not be recognized by the repository until you add it to the control tables.

To add a privilege to the control tables

1. In the EXTEND dialog, choose VIEW, TYPE. This gives you access the PRIV entity type.
2. Choose VIEW, LIST, ENTITY. This enables you to retrieve the desired privilege definition.
3. Choose SYSTEM, EXTEND, BUILD to add the privilege to the control tables. CA Repository displays a message indicating the success of the build.

Modify a Privilege

As your requirements change, it may be necessary to modify certain aspects of a privilege. Use the following procedure to modify and update a privilege.

To modify a privilege, do the following from the CA Repository

1. Choose VIEW, DIALOG. This gives you access the EXTEND dialog.
2. Choose VIEW, TYPE. This gives you access the PRIV entity type.
3. Choose VIEW, LIST, ENTITY, then select the PRIV instance you would like to modify.

   Note: If the PRIV instance is no longer in the Extend model, you can import it from the control tables. For more information, see Extract Altered Privilege Data.
4. If you need to change the PRIV instance itself, make the appropriate changes and choose EDIT, UPDATE. This commits the modifications.
5. If you need to change which commands the privilege can access:
   a. Choose EDIT, GOTO to select the PRIV CMD association type. CA Repository automatically displays all associations that use the PRIV instance, up to the maximum number that it can display at one time.
   b. Insert or delete PRIV CMD associations as necessary to link the privilege to the appropriate commands, then choose EDIT, GOTO to redisplay the PRIV entity type and reselect the PRIV instance.

6. If you need to change which dialogs, entity types, statuses, and workstations the privilege can access, repeat Step 5 for any DLG PRIV, PRIV STS, PRIV ENT, or WORKPRIV associations.
7. Choose SYSTEM, EXTEND, BUILD to rebuild the PRIV entity. This command replaces the privilege and all its associated objects in the control tables with the modified versions.

Extract Altered Privilege Data

If the privilege data in the Extend model no longer exists or was altered, you can use the IMPORT command to bring data directly from the control tables to the Extend model storage tables.

To import data from control tables to storage tables

1. Choose VIEW.DIALOG to select the EXTEND dialog.
2. Choose VIEW.TYPE to select the PRIV entity type.

   Note: To add or remove any data from the control tables that pertains to:

   - Privileges (PRIV, DLG PRIV, PRIV ENT, PRIV STS, and PRIV CMD), always use the PRIV entity type
   - Workstation privileges (WORKPRIV), you can use either the PRIV entity type or the ALL WORK entity set
3. Choose SYSTEM, EXTEND, IMPORT. A list of privilege definitions currently in the control tables appears.
4. Enter **S** in the SEL field of each privilege you want to import, then press Enter.
5. Enter a valid status for the EXTEND dialog at the prompt and press Enter.

   Note:

   - If the selected privilege is not in the repository model, a progress message appears stating that the import request is being processed.
   - If the privilege already exists in the repository model but was modified, the import progresses as explained above. However, the system overwrites the existing PRIV entity with a new one.
6. You can now use the online editing capabilities to select and make modifications to the PRIV instance and its associated instances as necessary.

IMPORTANT! To avoid loss of associations and relationships, make sure that any required COMMAND, DIALOG, META ENT, STATUS, and ALL WORK entities are in the Repository before importing a privilege.

Remove a Privilege

When it is necessary to remove a privilege so that it no longer provides access to the repository for any users, you need to remove the appropriate PRIV instance from the repository control tables.

To remove a user from the control tables

1. From CA Repository, choose VIEW, DIALOG to select the EXTEND dialog.
2. Choose VIEW, TYPE to select the PRIV entity type.
3. Use SYSTEM, EXTEND, REMOVE to display a list of all user definitions currently in the control tables. This list looks similar to the Import list shown in the chapter "Managing Repository Users."
4. Enter **S** in the SEL field of each privilege you want to remove, then press ENTER.

   A message appears indicating that the privilege definitions were successfully removed from the control tables.

IMPORTANT! Selecting a privilege from the Remove window removes that privilege from the control tables and eliminates any associations of the privilege to commands, dialogs, entity types, statuses, or workstations. The REMOVE command will not remove or otherwise affect the data in the Repository data tables (the Extend model).

How to Set Up Default Privileges for New Users

In some cases you may find it advantageous to set up one or more **default privileges** that are available to *undefined* repository users.

Undefined users have Logon IDs that are not associated with a USER instance. For more information, see the chapter "Managing Repository Users." Default privileges can simplify privilege and user administration at sites that have a large number of users or a user base that frequently changes.

Setting up default privileges consists of two basic procedures:

- Defining and building the privileges that you want to provide to undefined users
- Activating the Default Privilege feature

Define Default Privileges

Like all repository privileges, default privileges are defined as occurrences of the PRIV entity type.

Associations that link these privilege definitions to command, dialog, entity type, status, and workstation definitions determine what authority a particular privilege provides. For more information, see the Add Privileges earlier in this chapter.

Activate the Default Privilege Feature

In order for the Default Privilege feature to work, the CA Repository installer must have granted EXEC to PUBLIC on the repository plans. For more information, see the Bind the Utility Plans and Packages step in the "Installing for the First Time" chapter or the "Upgrading to the Current Release" chapter in the *CA Repository for z/OS Installation Guide*.

To activate the default privilege

1. Start the CA Repository and open the ARZOS CLIST. Look for the following.

```
/*  THIS PROGRAM CALL WILL ADD DEFAULT PRIVILEGES FOR A USER
/*  TO ACTIVATE THIS FEATURE JUST UNCOMMENT THE CALL
/* AND UPDATE THE "DEF Priv" NAMES TO THE DEFAULT PRIVILEGES YOU
/*      WANT ASSIGNED
/*  ISPEXEC SELECT PGM(DBXPVAD) NEWAPPL(DBEX) PASSLIB +
        PARM('/N,&ZUSER.,DEF PRIV1,DEF PRIV2,DEF PRIV3'
/*  END OF DEFAULT PRIVILEGE
```

2. Delete the comment characters in the lines starting with ISPEXEC and PARM.
3. Replace the DEF PRIV1, DEF PRIV2, and DEF PRIV3 parameters with the names of existing privileges (PRIV entities) that will be default privileges. You must specify at least one and up to three PRIV entities.

Command Checklist

The following table lists the commands available within CA Repository. To use this table, place a check mark beside each command to create a list of the commands you want to be available through a particular privilege.

When you create new privileges:

- Use a separate command list for each privilege
- Check off the commands to be part of that privilege
- If a command will be restricted to a specific status or entity type, be sure to note it on the list

	Window	Command
☐	BLOAD (Batch Load)	BLOAD.NOVALID
☐	COLUMN	OPERATOR
☐	COMMAND (Command Profile)	CLEAR
☐	COMMAND	DELETE
☐	COMMAND	RESELECT
☐	COMMAND	SAVE
☐	DBEXCEL (Edit Window)	EDIT.CHGSCR.CHANGE
☐	DBEXCEL	EDIT.CHGSCR.ERASE
☐	DBEXCEL	EDIT.CHGSCR.RESEQ
☐	DBEXCEL	EDIT.CHGSCR.SETVER
☐	DBEXCEL	EDIT.COPY
☐	DBEXCEL	EDIT.COPY.ALL
☐	DBEXCEL	EDIT.COPY.OTHER
☐	DBEXCEL	EDIT.COPY.RELATED
☐	DBEXCEL	EDIT.COPY.TEXT
☐	DBEXCEL	EDIT.DELETE
☐	DBEXCEL	EDIT.DOMAIN
☐	DBEXCEL	EDIT.NAVIGATE.GOTO
☐	DBEXCEL	EDIT.NAVIGATE.JUMP
☐	DBEXCEL	EDIT.INSERT
☐	DBEXCEL	EDIT.LOCK.ADD

	Window	Command
☐	DBEXCEL	EDIT.LOCK.SHOW
☐	DBEXCEL	EDIT.LOCK.REMOVE
☐	DBEXCEL	EDIT.LOCK.DELETE
☐	DBEXCEL	EDIT.MINIEDIT
☐	DBEXCEL	EDIT.MGRATON.MIGRATE
☐	DBEXCEL	EDIT.MGRATION.MIGTEST
☐	DBEXCEL	EDIT.MGRATION.MIGRPDEL
☐	DBEXCEL	EDIT.MGRATION.MIGBATCH
☐	DBEXCEL	EDIT.MGRATION.MIGCOPY
☐	DBEXCEL	EDIT.SELECT
☐	DBEXCEL	EDIT.SPECIAL.CHGTYPE
☐	DBEXCEL	EDIT.SPECIAL.MERGE
☐	DBEXCEL	EDIT.SPECIAL.REPLACE
☐	DBEXCEL	EDIT.SPECIAL.PATHDEL
☐	DBEXCEL	EDIT.SPECIAL.PATHADD
☐	DBEXCEL	EDIT.SYNC
☐	DBEXCEL	EDIT.TEXT.DESCRIPT
☐	DBEXCEL	EDIT.TEXT.COMMENT
☐	DBEXCEL	EDIT.TEXT.TEXTBLC
☐	DBEXCEL	EDIT.TEXT.IEWPICT
☐	DBEXCEL	EDIT.VIEW
☐	DBEXCEL	EDIT.UPDATE
☐	DBEXCEL	FILE.EXPORT...
☐	DBEXCEL	FILE.IMPORT...
☐	DBEXCEL	OPTIONS.DDLDB2
☐	DBEXCEL	OPTIONS.DDLIMS
☐	DBEXCEL	OPTIONS.OFFSET
☐	DBEXCEL	OPTIONS.QUICK.DB2
☐	DBEXCEL	OPTIONS.QUICK.DBD
☐	DBEXCEL	OPTIONS.QUICK.MAP

	Window	Command
☐	DBEXCEL	OPSTIONS QUICK.PSB
☐	DBEXCEL	OPTIONS.QUICK.REC
☐	DBEXCEL	OPTIONS.QUICK.SEGMENT
☐	DBEXCEL	OPTIONS.QUICK.88
☐	DBEXCEL	OPTIONS.CATSYNC.SYNCDB2
☐	DBEXCEL	OPTIONS.CATSYNC.SYNCDEL
☐	DBEXCEL	OPTIONS.CATSYNC.SYNCTEST
☐	DBEXCEL	OPTIONS.COPYIN.DBD
☐	DBEXCEL	OPTIONS.COPYIN.PSB
☐	DBEXCEL	OPTIONS.COPYIN.RECORD
☐	DBEXCEL	OPTIONS.REPORTS.(Report Type)
☐	DBEXCEL	OPTIONS.TOOLS.BMC
☐	DBEXCEL	OPTIONS.TOOLS.PLATINUM
☐	DBEXCEL	OPTIONS.TOOLS.QMF
☐	DBEXCEL	OPTIONS.USERRPT.ADDTORPT
☐	DBEXCEL	OPTIONS.USERRPT.DELETETB
☐	DBEXCEL	OPTIONS.WORKSTN.ADD
☐	DBEXCEL	OPTIONS.WORKSTN.ADDONLY
☐	DBEXCEL	OPTIONS.WORKSTN.DELETE
☐	DBEXCEL	OPTIONS.WORKSTN.DELONLY
☐	DBEXCEL	SYSTEM.COMMANDS
☐	DBEXCEL	SYSTEM.CRITERIA
☐	DBEXCEL	SYSTEM.DEFAULTS
☐	DBEXCEL	SYSTEM.LOGO
☐	DBEXCEL	SYSTEM.EXTEND.BUILD
☐	DBEXCEL	SYSTEM.EXTEND.CREATE
☐	DBEXCEL	SYSTEM.EXTEND.COMPSTAT
☐	DBEXCEL	SYSTEM.EXTEND.GRANT
☐	DBEXCEL	SYSTEM.EXTEND.IMPORT
☐	DBEXCEL	SYSTEM.EXTEND.NEXTID

	Window	Command
☐	DBEXCEL	SYSTEM.EXTEND.RECREATE
☐	DBEXCEL	SYSTEM.EXTEND.REMOVE
☐	DBEXCEL	SYSTEM.EXTEND.REVOKE
☐	DBEXCEL	SYSTEM.EXTEND.STATMAP
☐	DBEXCEL	SYSTEM.EXTEND.VIEWMSG
☐	DBEXCEL	VIEW.CURRMAP
☐	DBEXCEL	VIEW.DIALOG
☐	DBEXCEL	VIEW.IMPACT.USES
☐	DBEXCEL	VIEW.IMPACT.WHERE
☐	DBEXCEL	VIEW.IMPACT.XREF
☐	DBEXCEL	VIEW.LIST.ENTITY
☐	DBEXCEL	VIEW.LIST.SOURCE
☐	DBEXCEL	VIEW.LIST.TARGET
☐	DBEXCEL	VIEW.MAINMENU
☐	DBEXCEL	VIEW.MODE
☐	DBEXCEL	VIEW.QUEUE.ON
☐	DBEXCEL	VIEW.QUEUE.OFF
☐	DBEXCEL	VIEW.QUEUE.CLEAR
☐	DBEXCEL	VIEW.QUEUE.ENTITY
☐	DBEXCEL	VIEW.QUEUE.SOURCE
☐	DBEXCEL	VIEW.QUEUE.TARGET
☐	DBEXCEL	VIEW.TYPE
☐	DBEXCEL	PROFILE.CMDLINE
☐	DBEXCEL	PROFILE.COMMANDS
☐	DBEXCEL	PROFILE.DEFAULTS
☐	DB2MGR (DB2 DDL Manager)	MAINTAIN.ALTER
☐	DB2MGR (DB2 DDL Manager)	MAINTAIN.CREATE
☐	DB2MGR (DB2 DDL Manager)	MAINTAIN.DROP
☐	DB2MGR (DB2 DDL Manager)	MAINTAIN.LOAD
☐	DB2MGR (DB2 DDL Manager)	MAINTAIN.UNLOAD

	Window	Command
☐	DB2MGR (DB2 DDL Manager)	UTILS.COPY
☐	DB2MGR (DB2 DDL Manager)	UTILS.MERGCOPY
☐	DB2MGR (DB2 DDL Manager)	UTILS.REORG
☐	DB2MGR (DB2 DDL Manager)	UTILS.RUNSTAT
☐	DB2MGR (DB2 DDL Manager)	UTILS.STOSPACE
☐	DB2MGR (DB2 DDL Manager)	UTILS.UNLOAD
☐	DEFAULTS (Default Profile)	CLEAR
☐	DEFAULTS (Default Profile)	DELETE
☐	DEFAULTS (Default Profile)	RESELECT
☐	DEFAULTS (Default Profile)	SAVE
☐	FUNCTION (Function Window)	COLUMN
☐	FUNCTION (Function Window)	VALUE
☐	IMSMGR (IMS DDL Manager)	MAINTAIN.BUILD
☐	IMSMGR (IMS DDL Manager)	MAINTAIN.DELETE
☐	IMSMGR (IMS DDL Manager)	UTILS.UTIL1
☐	LIST	BROWSE
☐	LIST	CRITERIA
☐	LIST	IMPACT.USES
☐	LIST	IMPACT.WHERE
☐	LIST	IMPACT.XREF
☐	LIST	TEXT.DESCRIPT
☐	OPERATOR	COLUMN
☐	OPERATOR	FUNCTION
☐	OPERATOR	VALUE
☐	PRINT	BATCH
☐	PRINT	SCREEN
☐	SEARCH (Search Criteria)	CLEAR
☐	SEARCH (Search Criteria)	QUERY
☐	SEARCH (Search Criteria)	QUERY.DELETE
☐	SEARCH (Search Criteria)	QUERY.SAVE

	Window	Command
☐	SEARCH (Search Criteria)	QUERY.SAVEALL
☐	SEARCH (Search Criteria)	QUERY.SELECT
☐	SEARCH (Search Criteria)	SQL.COLUMN
☐	SEARCH (Search Criteria)	SQL.FUNCTION
☐	SEARCH (Search Criteria)	SQL.OPERATOR
☐	SEARCH (Search Criteria)	SQL.VALUE
☐	TBLIST (Table List)	TBLLIST.CRITERIA
☐	TBLIST (Table List)	TBLLIST.HELP
☐	TSEARCH (Table List Search)	TSEARCH.CLEAR
☐	TSEARCH (Table List Search)	TSEARCH.HELP
☐	TSEARCH (Table List Search)	TSEARCH.SQL
☐	TSEARCH (Table List Search)	TSEARCH.SQL.COLUMN
☐	TSEARCH (Table List Search)	TSEARCH.SQL.FUNCTION
☐	TSEARCH (Table List Search)	TSEARCH.SQL.OPERATOR
☐	VALUE	OPERATOR
☐	XREF (Cross Reference)	BROWSE
☐	XREF (Cross Reference)	IMPACT
☐	XREF (Cross Reference)	IMPACT.USES
☐	XREF (Cross Reference)	IMPACT.WHERE
☐	XREF (Cross Reference)	IMPACT.XREF
☐	XREF (Cross Reference)	TEXT.DESCRIPT

Chapter 5: Working with Statuses

This chapter explains how to add, modify, and remove statuses and migration paths.

This section contains the following topics:

Repository Status and Level Numbers (see page 71)
Add a New Status to the Repository (see page 75)
Modify Statuses and Migration Paths (see page 80)
Remove a Status or Migration Path (see page 82)
Statuses That Support Projects (see page 83)

Repository Status and Level Numbers

Status is used to indicate, control and limit:

- The project to which a specified entity belongs
- The current position of an entity in the application life cycle.
- The current phase of an entity in an application development life cycle

  This is augmented by:

  - The level number of the status itself
  - The Repository migration path relationships
  - An entity's movement between life cycle phases
- Access to repository data based on the projects, system phases or application lifecycle phases

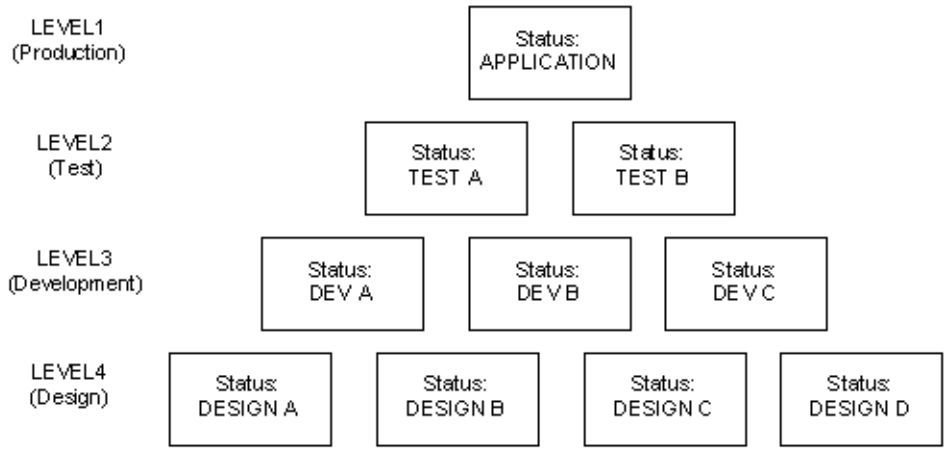
Status Level Numbers

Level numbers:

- Are assigned to statuses
- Control the directions that relationships and associations can point for their source and target requirements
- Control the direction an entity can be updated

Statuses with lower level numbers attach to entities in later phases of the application development life cycle. Statuses with higher-level numbers are used for earlier or developmental entities.

The following diagram shows statuses in a simplified application life cycle.



Possible Status Levels

Level Number Restrictions

When you define relationships:

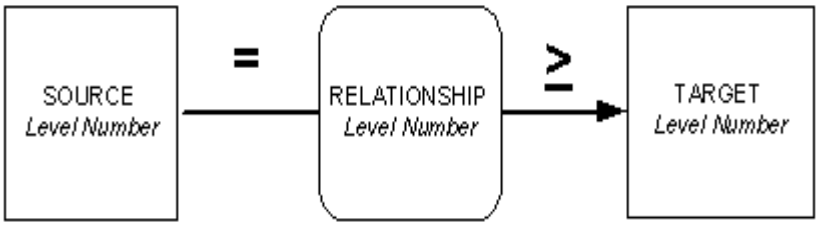
- The status level number of the relationship and its source entity must be equal.
- The target can have a status level number that is equal to or less than the relationship.

  For example, if you want to define a relationship with a status level number of 2, then:

  – The relationship's source entity must also have a status level number of 2

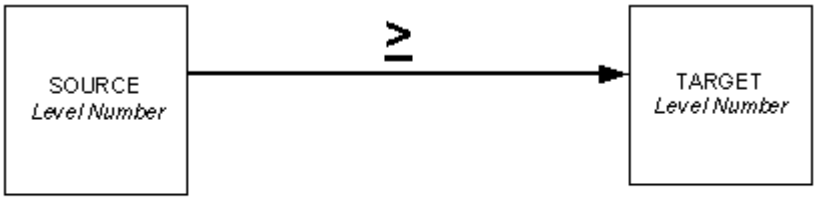
  – The relationship's target entity must have a status level number less than or equal to 2

The following diagram shows the relationship between the level numbers.



Level Numbers Relationship

When you define associations, the status level of the target must be less than or equal to the status level of the source.



Source and Target Status Levels

Migration Paths

Migration paths:

- Are defined as instances of the relationship type MIG PATH
- Determine the levels to which an entity can move

Use Migration paths to:

- Control entities in an application life cycle
- Prevent an entity at a test level from being inadvertently moved back to a design level
- Allow entities to be moved to a production level

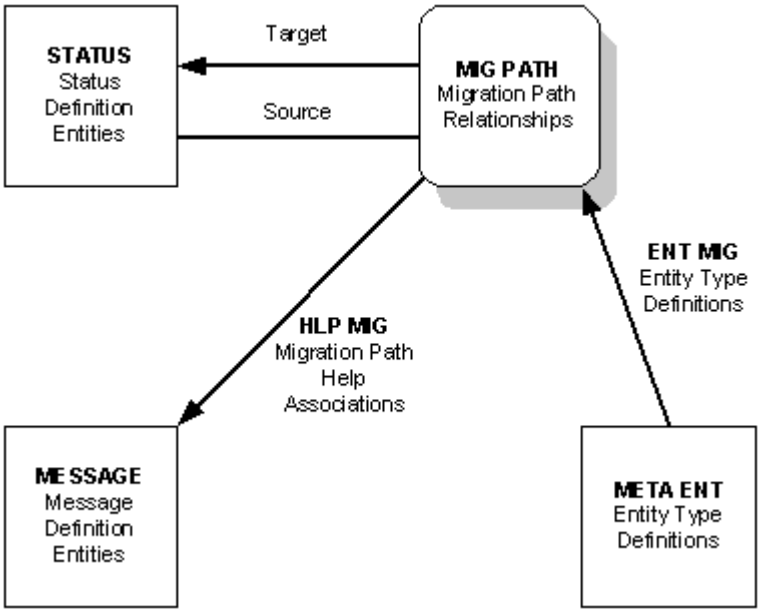
When you change from one status to another, you must determine which status is the source and which is the target. Only by determining the source and target can you determine the acceptable direction of movement when the source status is the From Status and the target status is the To Status.

The Status Model

The Status Model contains the following Extend model components:

- STATUS entity type
- MIG PATH relationship type
- HLP MIG and ENT MIG association types

The following diagram represents the Status Model.



Note: The MESSAGE and META ENT entity types are not direct parts of the Status Model. They are included only to show their relationship to the associations that bind them to the Migration Path relationship type.

The following table provides you with the descriptions of the previous components:

Component	Description
STATUS	Defines statuses. It is an entity type.
MIG PATH	Defines paths that are used when changing an entity from one status to another. It is a relationship type.
HLP MIG	Joins MIG PATH relationships (migration paths) to existing MESSAGE entities (Help messages) to provide informative messages about a particular migration path. It is an association type.
ENT MIG	Determines the migration paths that are valid for a particular entity type. It is an association type.

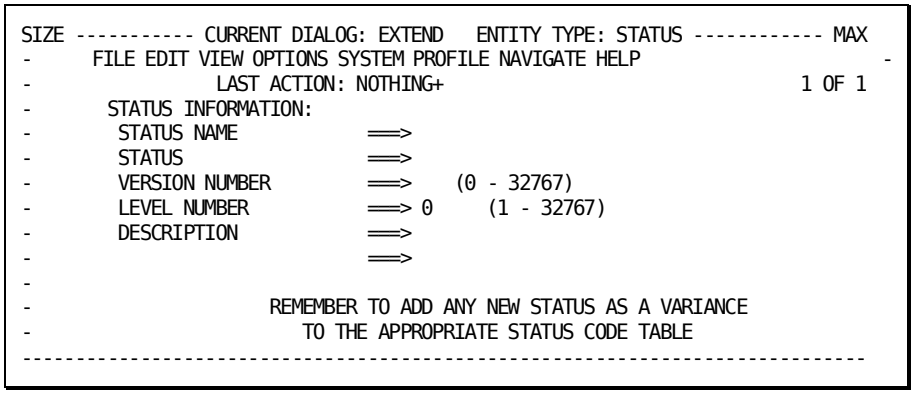
To open an Edit screen for a Status Model component

1. From CA Repository choose VIEW, DIALOG to access the EXTEND dialog.
2. Choose VIEW, TYPE to open an Edit screen for any of the components listed in the table.

Add a New Status to the Repository

To add a new status to the repository

1. From the CA Repository, choose VIEW, DIALOG to access the EXTEND dialog.
2. Choose VIEW, TYPE to access the STATUS entity type; the STATUS Edit panel appears.



3. Enter values into the STATUS input fields as follows:

Field	Enter
STATUS NAME	A name for the status you want to add.
STATUS	The status to which this status belongs.
VERSION NUMBER	A version number.
LEVEL NUMBER	A status level number. **Note:** For more information, see Understanding Status Level Numbers, earlier in this chapter.
DESCRIPTION	A description of the status.

4. Choose EDIT, INSERT to add the STATUS entity to the Repository.
5. Choose SYSTEM, EXTEND, BUILD to add the status to the control tables. A message appears indicating the success of the build.

Validate Statuses

Any newly created status must be added to the appropriate status code tables before it can be used by entity types that use those tables.

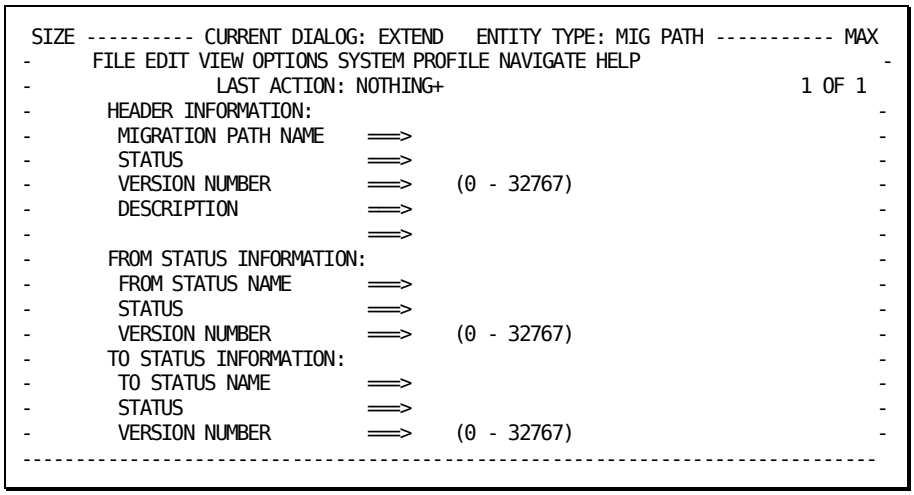
Note: The control data shipped with CA Repository uses code table E29 for EXTEND dialog statuses and code table 024 for all others.

For information about adding newly created statuses to status code tables, see the "Working with Code Tables" chapter.

Add a Migration Path

To add a migration path

1. From CA Repository, choose VIEW, DIALOG to access the EXTEND dialog.
2. Choose VIEW, TYPE to access the MIG PATH entity type. The MIG PATH edit screen appears.



3. Enter values into the fields as follows:

Field	Enter
Migration Path Name	The name of the MIG PATH.
Status	The status of the MIG PATH.
Version Number	The version number of the MIG PATH.
Description	The description of the MIG PATH.
From Status Name	The name of the FROM STATUS.

Field	Enter
Status	The status of the FROM STATUS.
Version Number	The version number of the FROM STATUS.
To Status Name	Specifies the name of the TO STATUS.
Status	The status of the TO STATUS.
Version Number	The version number of the TO STATUS.

4. Choose EDIT, INSERT to add the MIG PATH entity to the Repository.

Limit the Migration Path to Specific Entity Types

To limit the migration path to specific entity types

1. From the CA Repository, choose VIEW, DIALOG to access the EXTEND dialog.
2. Choose VIEW, TYPE to access the ENT MIG association type. The ENT MIG Edit screen appears.

```
 SIZE ----------- CURRENT DIALOG: EXTEND   ENTITY TYPE: ENT MIG ----------- MAX
-     FILE EDIT VIEW OPTIONS SYSTEM PROFILE NAVIGATE HELP                     -
-                LAST ACTION: NOTHING+                              1 OF 1
-       META ENT INFORMATION:                                                -
-        META ENT NAME          ===>                                         -
-        STATUS                 ===>                                         -
-        VERSION NUMBER         ===>    (0 - 32767)                          -
-       PATH INFORMATION:                                                    -
-        PATH NAME              ===>                                         -
-        STATUS                 ===>                                         -
-        VERSION NUMBER         ===>    (0 - 32767)                          -
 ------------------------------------------------------------------------------
```

3. Enter values into the ENT MIG input fields as follows:

Field	Enter
META ENT NAME	The name of the META ENT.
STATUS	The status of the META ENT.
VERSION NUMBER	The version number of the META ENT.
PATH NAME	The name of the MIG PATH.
STATUS	The status of the MIG PATH.
VERSION NUMBER	The version number of the MIG PATH.

Note: If no associations are created, the migration path is available for all entity types. However, if you want to limit the migration path to one or more entity types, you need to define and insert a separate association that links the new migration path entity to each of the selected entity types.

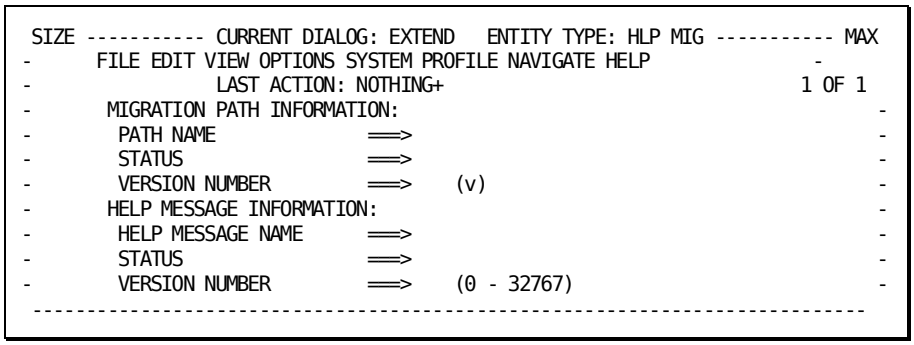
4. Choose EDIT, INSERT to update the Repository.

Provide Migration Path Help Messages

Providing migration path help messages is not required, but providing help is recommended. Use the following procedure to provide migration path help messages.

To provide migration path help messages

1. From CA Repository, use VIEW, DIALOG to access the EXTEND dialog.
2. Choose VIEW, TYPE to access the HLP MIG association type. The HLP MIG Edit screen appears.



3. Enter values into the ENT MIG input fields as follows:

Field	Enter
Path Name	The name of the MIG PATH.
Status	The status of the MIG PATH.
Version Number	The version number of the MIG PATH.
Help Message Name	The name of the MESSAGE.
Status	The status of the MESSAGE.
Version Number	The version number of the MESSAGE.

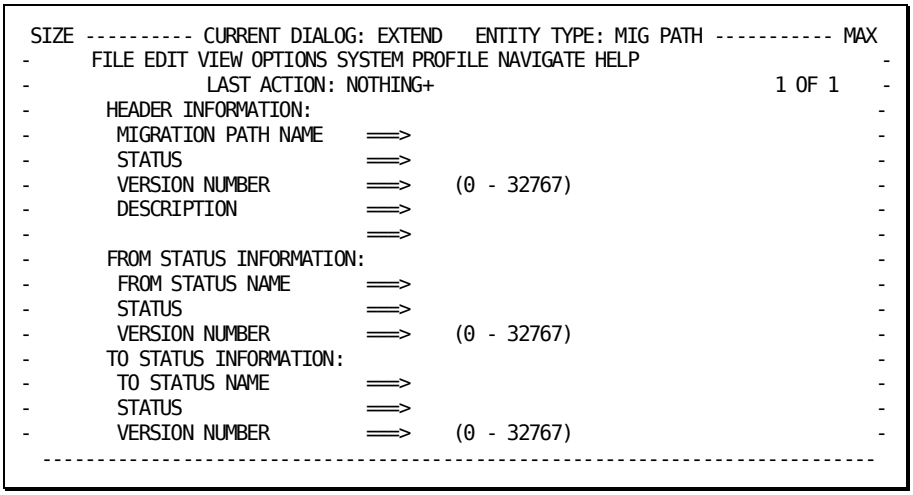
4. Use the EDIT, INSERT command to update the Repository.

   Note: Although this association links a help message to a migration path, it will not provide the specified message unless the message is first defined and added to the control tables. Message administration is discussed in detail in the "Customizing Messages" chapter of this guide.

Add a Migration Path to Control Tables

To add the migration path to control tables

1. From CA Repository, choose VIEW, DIALOG to access the EXTEND dialog.
2. Choose VIEW, TYPE to access the MIG PATH association type. The MIG PATH Edit screen appears.



3. Select the MIG PATH relationship that you want to add to the control tables and Choose SYSTEM, EXTEND, BUILD to add the MIG PATH relationship to the control tables.

   This adds the migration path to the control tables as well as the ENT MIG and HLP MIG associations. A message appears indicating the success or failure of the build. If successful, the new path can be used when updating entities.

Modify Statuses and Migration Paths

As your site's requirements change, it may be necessary to modify certain aspects of a status or migration path. Use the following procedure to modify a status or migration path.

To modify a migration path or status

1. From the CA Repository, choose VIEW, DIALOG to access the EXTEND dialog.
2. Choose VIEW, TYPE to access MIG PATH or STATUS entity type. The MIG PATH or STATUS Edit screen appears.
3. Select the desired MIG PATH association or STATUS entity.
4. Make modifications as necessary.
5. Choose EDIT, UPDATE to update the repository.
6. Use SYSTEM, EXTEND, BUILD to replace the MIG PATH association or STATUS entity and all their related components in the control tables with the modified versions. A message indicating the success of the build appears.

Extract Data from Control Tables to Extend Data Tables

If the data in the Repository model no longer exists or has been altered, you can use the IMPORT command to bring data directly from the control tables to the Extend data tables. Use the following procedure to import data from control tables to storage tables.

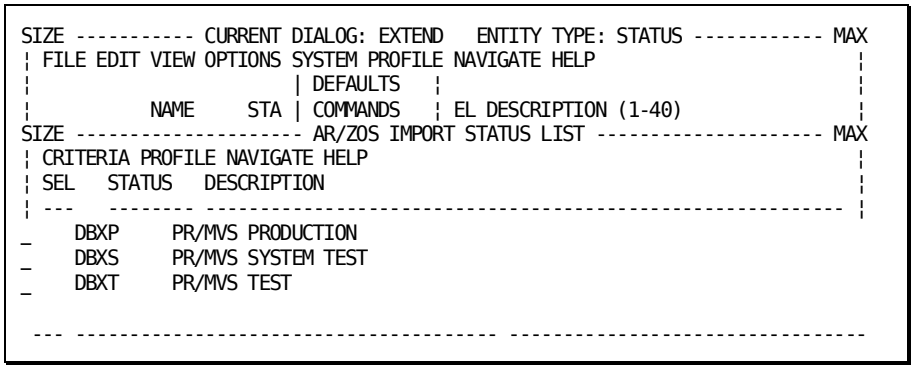
Important! To avoid loss of associations and relationships, make sure that any required META ENT, MESSAGE and STATUS entities are in the Repository before importing a status or migration path. Imports should be performed in the following order:

1 - messages, 2 - entities, 3 - statuses, and 4 - migration paths.

To import data from control tables to storage tables

1. From the CA Repository, choose VIEW, DIALOG, to access the EXTEND dialog.
2. Choose VIEW, TYPE to access the MIG PATH or STATUS entity types. The MIG PATH or STATUS Edit screen appears.

3. Execute SYSTEM, EXTEND, IMPORT. An Import screen containing a list of the statuses (or migration paths) currently in the control tables appears, similar to the one that follows.



4. Enter **S** for select next to the statuses or migration paths you want to import; then press Enter.

   Notes:

   - If the selected status or migration path is not in the Repository model, a progress message appears stating that the import request is being processed.

     Processing is very rapid, so the progress message may appear and disappear before you can read it. Once the process message disappears, another message appears stating that the import completed.

   - If the status or migration path already exists in the repository model but it was modified, the import progresses as previously explained-however, the system adds a new STATUS entity/MIG PATH relationship with a higher version number than its pre-existing counterpart.

5. Make modifications to the status, migration path, or both definitions as necessary.

6. Use the appropriate Edit command to update the Repository with your modifications.

7. Choose SYSTEM, EXTEND, BUILD to return the altered status or migration path definition to the control tables.

Remove a Status or Migration Path

Use the following procedure to remove a status or migration path.

Important! Selecting a status or migration path from the Remove window will remove that status or migration path from the control tables and eliminate any associations of the status or migration path to commands, dialogs, entity types, or statuses. The REMOVE command will not remove or otherwise affect the data in the Repository data tables (the Extend model).

To remove a status or migration path

1. From CA Repository, choose VIEW, DIALOG to access the EXTEND dialog.
2. Choose VIEW, TYPE to access the MIG PATH or STATUS entity types.
3. Execute SYSTEM, EXTEND, REMOVE. A list of all the entities currently in the control tables for the selected entity type appears in the REMOVE screen. It looks similar to the following example.

```
SIZE ----------- CURRENT DIALOG: EXTEND   ENTITY TYPE: STATUS ---------- MAX
-      FILE EDIT VIEW OPTIONS SYSTEM PROFILE NAVIGATE HELP              -
-                LAST ACTION | DEFAULTS   -                            1 OF 1
-       STATUS INFORMATION:   | COMMANDS   -                           -
SIZE -------------------- AR/ZOS REMOVE STATUS LIST ---------- ------- MAX
-      CRITERIA PROFILE NAVIGATE HELP                                    *
-      SEL   STATUS   DESCRIPTION                                        *
-      ---   -------- ------------------------------------------------- *
-        _    DBXP     PR/MVS PRODUCTION                                 *
-        _    DBXS     PR/MVS SYSTEM TEST                                *
-        _    DBXT     PR/MVS TEST                                       *
-        _    DB2P     PRODUCTION DB2                                    *
-        _    DB2S     SYSTEM TEST DB2                                   *
-        _    DB2T     TEST DB2                                          *
-        _    DDLP     PRODUCTION DDL                                    *
-        _    DDLS     SYSTEM TEST DDL                                   *
-        _    DDLT     TEST DDL                                          *
-        _    EXTP     EXTEND PRODUCTION                                 *
-        _    EXTS     EXTEND SYSTEM TEST                                *
-        _    EXTT     EXTEND TEST                                       *
 ***********************************************************
```

4. Enter an **S** for select next to each status or migration path you want to remove, and then press Enter. A message appears indicating that status or migration path definitions were successfully removed from the control tables.

Statuses That Support Projects

The following is a list of the status types you can use to support projects. The sections that follow explain each status in detail.

Status	Project
PROD	Production
TEST	Test
DEV01	Development Project 1
DEV02	Development Project 2

Production Status (PROD)

The production-level status (PROD) contains a copy of each system component considered to be at a *finished* status. Multiple copies of a single component are required in production, such as different attributes, relationships, or associations. Any project that uses an existing entity definition can establish a relationship to that definition.

For example, if you add a table in TEST, you can use existing production elements when you define columns without having to make a new copy of those entities.

Although entities in a production status cannot reference entities in a test or development status, entities in test or development can use entities in production. This rule ensures that production items are not affected when test or developmental items change.

Test Status

Test status components are viewed as integration items. When you move an item from development to test, no corresponding item is left in development. This avoids redundancy and discrepancy and forces management of integration issues.

The TEST status contains system components that are:

- Not at a production level
- Different from their production versions
- Not viewed as project-controlled components

Separating test and project statuses enables you to resolve integration issues and integrate testing. Although entities at a test-level can not use development-level entities as their target, entities in development can use entities in a test-level.

Development Status (DEVnn)

A separate development status must be created for each project. This includes new items added by a project team and copies made of test or production items.

Chapter 6: Working with Commands

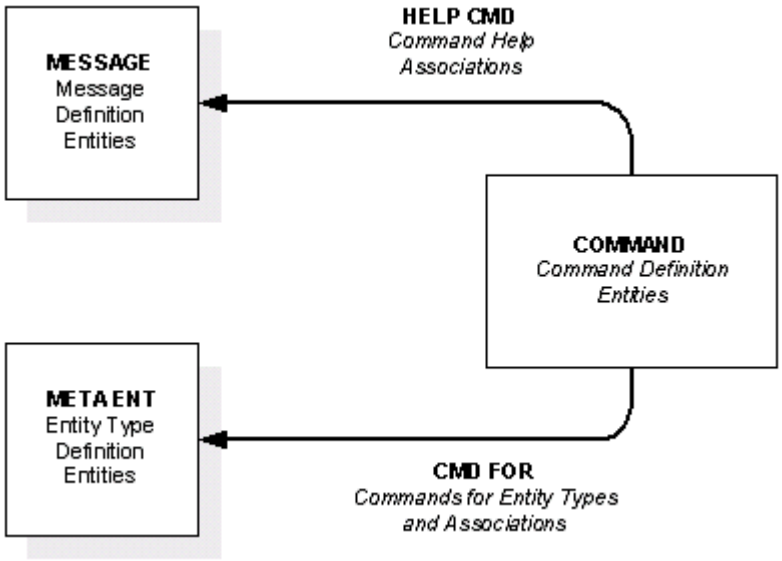
This chapter explains how to add and modify CA Repository commands using the Extend model.

This section contains the following topics:

The Command Model (see page 85)
Define, Link, and Add a Command (see page 86)
Modify a Repository Command (see page 91)
Remove a Repository Command (see page 92)

The Command Model

The following illustration shows the Extend model object types that the repository uses to define menus and commands.



The purpose of the objects is listed in the following table.

Object	Purpose
COMMAND	Defines CA Repository commands
CMD FOR	Links command definitions to META ENT definitions to limit the command to specific entity types

Object	Purpose
HLP CMD	Links command definitions to CA Repository help message definitions Messages joined to a command through this association type are displayed when a user types a question mark over a command on a pull-down menu.
MESSAGE	Defines CA Repository error and help messages
META ENT	Defines CA Repository object types

Note: MESSAGE and META ENT have shadows in the figure to indicate that, while used as part of a command definition, they are not actually part of the command model. Therefore, they should not be built using the system extend build function without creating objects in the message model and entity models respectively. The message object is shown in the illustration because it is required to build a HLP CMD relationship. The META ENT object is shown because it is required to build a CMD FOR relationship.

META ENT is used when defining CA Repository entity types, while MESSAGE is used when defining CA Repository messages, see the "Customizing Messages" chapter later in this guide.

Define, Link, and Add a Command

The list that follows presents the steps required to add a new command. Each of these steps is explained in detail in the sections that follow.

To add a new command to CA Repository

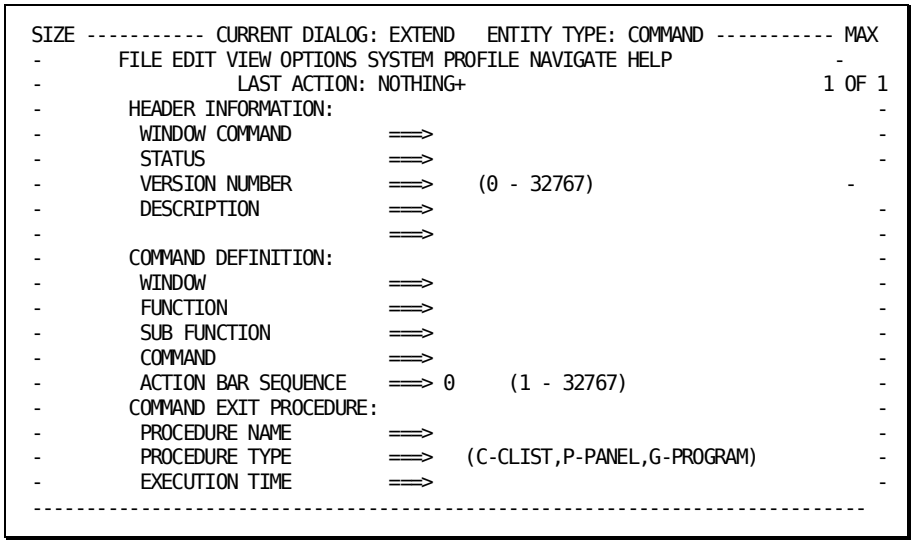
1. Define a command instance.
2. Link the command to a help message.
3. Link the command to one or more entity type definitions.
4. Add the command to the control tables.

Define a Command Instance

The first thing you must do when setting up a new CA Repository command is to insert a command instance that defines the command. The command entity type has attributes that allow you to specify in which window the command appears, as well as under which menu or submenu it will appear.

To define a command instance

1. From CA Repository, choose VIEW, DIALOG to open the EXTEND dialog.
2. Choose VIEW.TYPE to access the command entity type; the COMMAND Edit screen appears.



3. Enter values into the COMMAND fields as follows:

Field	Enter
Window Command	A name for the COMMAND instance
Status	A status for the COMMAND instance
Version Number	A version number for the COMMAND instance
Description	A description of the command
Window	The name of the window in which you want the command to appear
Function	The function (on the menu bar) under which you want the command to appear
Sub Function	The sub function (submenu) under which you want the command to appear (if any)
Command	The name of the command as you want it to appear in the CA Repository

Field	Enter
	menu
Action Bar Sequence	The sequence number of the command within the menu bar or pull-down menu You will need to know the sequence number of other commands (if any) on the appropriate menu to determine how to number your new command.
Procedure Name	The name of the procedure the command is to run.
Procedure Type	Enter one of the following: C-If the procedure is a CLIST. P-If the procedure is a panel. G-If the procedure is a program.
Execution Time	B-Before A-After BA-Before/After

4. When you have completed all of the COMMAND fields, choose EDIT, INSERT to save the command definition to the repository.

Provide Help for a Command

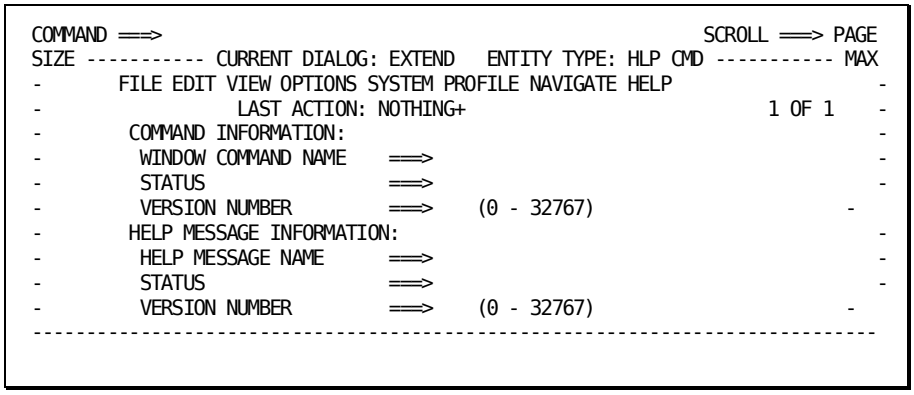
If you want to provide help for a command, you can link the command definition to a CA Repository message. CA Repository displays messages linked to commands when a user types a question mark over the command on a pull-down menu.

To link a command to a help message

1. From CA Repository, choose VIEW, DIALOG to access the EXTEND dialog.

   Note: If you already have the appropriate COMMAND instance displayed, use EDIT.GOTO to select the HLP CMD association type, then skip to Step 4.

2. Choose VIEW, TYPE to access the HLP CMD association type; the HLP CMD Edit screen appears.



3. Choose VIEW, LIST, SOURCE to select the COMMAND instance defining the command you want to link to a help message.
4. Choose VIEW, LIST, TARGET to select the MESSAGE instance defining a help message for your command.
5. Choose EDIT, INSERT to insert the new association into the repository.

Limit a Command to Specific Entity Types

By default, user-defined CA Repository commands are available for all object types. That is, you'll be able to select the command from the menu bar of any repository object type. If the command you are defining is only applicable to certain entity types, you may want to set it up so that it is only available in the appropriate Edit windows.

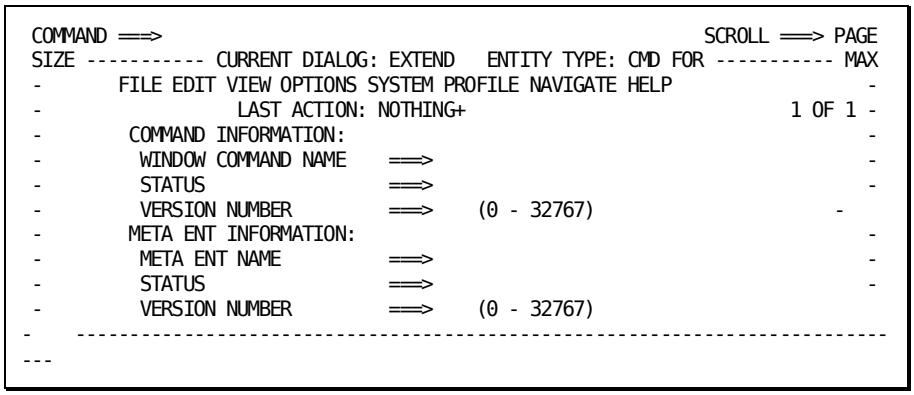
You can limit a command to specific entity types simply by creating links in the Extend model between the command definition (COMMAND) and the appropriate entity type definitions (META ENT). When you link a command to one or more entity types, CA Repository restricts the command to those types and disables it for all others. Disabled commands appear *dimmed* in the pull-down menus.

To limit commands to specific entity types

1. From CA Repository, choose VIEW, DIALOG to access the EXTEND dialog.

   Note: If you already have the appropriate COMMAND instance displayed, use EDIT.GOTO to select the CMD FOR association type, then skip to Step 4.

2. Choose VIEW, TYPE to access the CMD FOR association type; the CMD FOR Edit screen appears.



3. Choose VIEW, LIST, SOURCE to select the COMMAND instance defining the command you want to limit to certain entity types.
4. Choose VIEW, LIST, TARGET to select the META ENT instance defining an entity type that you want to be active.
5. Choose EDIT, INSERT to insert the new association into the repository.
6. If you want to enable the command for additional entity types, repeat Steps 3 through 5 to connect the command to any other META ENT definitions.

Add a Command to the Control Tables

After you define a command and link it to a help message and entity type, you need to add it to the control tables to activate it. Use the following procedure to add a command to the control tables.

To add a command to the control tables

1. From CA Repository, choose VIEW, DIALOG to access the Extend dialog.
2. Choose VIEW, TYPE to access the COMMAND entity type.

3. Choose VIEW, LIST, ENTITY to select the COMMAND instance representing the new command.
4. Choose SYSTEM, EXTEND, BUILD to add the command entity to the control tables. CA Repository automatically adds information in any related Command-model associations to the control tables with the new command. A message indicating the success or failure of the build appears after processing.

Modify a Repository Command

As your requirements change, it may be necessary to modify certain aspects of a command. Use the following procedure to modify and rebuild a command.

To modify a command

1. From CA Repository, choose VIEW, DIALOG to access the EXTEND dialog.
2. Choose VIEW, TYPE to access the COMMAND entity type. The COMMAND Edit screen appears.
3. Choose VIEW, LIST, ENTITY to select the desired COMMAND entity.
4. Make any necessary changes to the command definition and use EDIT.UPDATE to save the changes to the COMMAND instance.
5. If necessary, modify the HLP CMD linking the command definition to a help message.
6. If necessary, add or delete CMD FOR associations linking the command definition to entity type definitions (META ENT).
7. Reselect the COMMAND instance and choose SYSTEM, EXTEND, BUILD to replace the COMMAND entity and all its related components in the control tables with the modified versions. A message indicating the success of the build appears.

Extract Command Data from the Control Tables

If the command data in the Extend model no longer exists or has been altered, you can use the IMPORT command to bring data directly from the control tables to the Extend model storage tables.

To import data from control tables to storage tables

1. Choose VIEW, DIALOG to select the EXTEND dialog.
2. Choose VIEW, TYPE to select the COMMAND entity type.
3. Choose SYSTEM, EXTEND, IMPORT. A list of command definitions currently in the control tables appears.

4. Enter **S** in the SEL field of each command you want to import, then press Enter.

   Notes:

   - If the selected command is not in the repository model, a progress message appears stating that the import request is being processed. Processing is very rapid, so the progress message may appear and be removed before you can read it. Once the process message disappears, a message appears stating that the import was completed.
   - If the command already exists in the repository model but was modified, the import progresses as previously explained. However, the system adds a new COMMAND entity with a higher version number than its pre-existing counterpart.

5. You can now use the Editor to select and make modifications to the COMMAND instance and its associated instances as necessary.

WARNING! To avoid loss of associations and relationships, make sure that any required META ENT and MESSAGE entities are in the Repository before importing a command.

Remove a Repository Command

If you need to deactivate and CA Repository command, you can use the following procedure to remove the command definition from the control tables.

To remove a command

1. From CA Repository, choose VIEW, DIALOG to access the EXTEND dialog.
2. Choose VIEW, TYPE to access the COMMAND entity type.
3. Choose SYSTEM, EXTEND, REMOVE to display a list of all command definitions currently in the control tables.
4. Enter **S** next to each command you want to remove, then press Enter.

   A message appears indicating that a command definition was successfully removed from the control tables.

WARNING! Selecting a command from the Remove window will remove that command from the control tables, eliminating any associations between the command and other components. The REMOVE command does not affect the data in the CA Repository Extend model repository tables.

Chapter 7: Adding and Customizing Maps and Entity Types

This chapter discusses how to customize CA Repository maps and entities.

This section contains the following topics:

Repository Entity Types (see page 93)
The Quick Map Facility (see page 93)
Open the Quick Map Facility (see page 94)
Use the Quick Map Facility (see page 96)

Repository Entity Types

Repository entity types consist of three major parts:

- A name
- A map
- A dialog

Descriptions of these parts are provided in the following table.

Entity Type	Description
Name	The eight-character name used to access the entity type while in the repository
Map	A definition of the entity type's attribute types and screen literals
Dialog	A named group of logically related entity types to which the entity type belongs

All entity types in the repository are a unique combination of the parts described above. An entity type can use a different map in each of the dialogs to which it belongs.

The Quick Map Facility

Most of the steps required to build entity types and customize metadata models can be done using the Quick Map Facility. Quick Map adds the entity type/map combinations to the dialogs you specify by writing directly to control tables.

Use Quick Map to:

- Modify existing entity types.
- Create entity types from scratch.
- Use an existing entity type as a starting point for creating new entity types.

Note: You can still manually extend the repository using the Extend model.

Use TSO Edit Commands

The standard TSO edit commands can be used in the select area to manipulate the rows in the Quick Map window. The available Edit commands are:

Command	Action
I	Inserts a row
M	Moves a row Enter: A-To move a selected row After a particular row. B-To move a selected row Before a particular row.
D	Deletes a row

Note: A number after I, M, or D indicates the number of rows affected by the Edit command.

Open the Quick Map Facility

Use the following procedure to open the Quick Map Facility.

Before you begin:

- Choose SYSTEM, EXTEND, IMPORT to create a copy of the map you will manipulate. This allows you to rebuild the map in case of an error.

  or

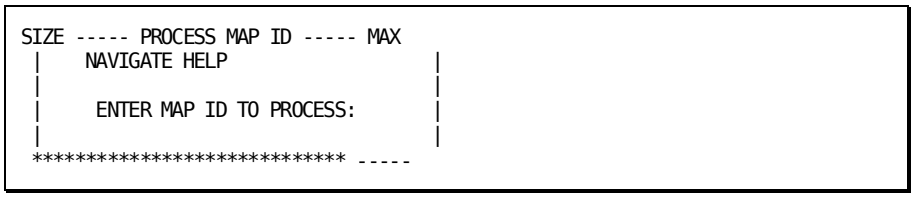
- Use Quick Map's SAVENEW command.

  This saves the current map with a new ID, leaving the original map preserved in the control tables. The original map can be removed once the new map is verified and tested.

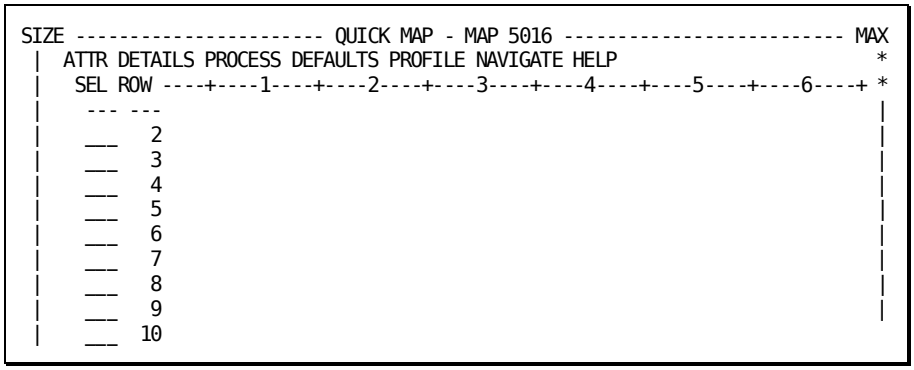
To open the Quick Map Facility

1. From CA Repository, choose VIEW, DIALOG to open a dialog.
2. Choose VIEW.TYPE to open an entity type.

   Note: You can access Quick Map from any dialog and entity type in CA Repository.
3. Choose OPTIONS, QUICK, MAP to access Quick Map. The Process Map ID prompt appears.



   Note: To find a Map ID and the entity type of the object you are currently viewing, execute the VIEW.CURRMAP command.
4. When building a new map, enter an unused Map ID; this displays the Quick Map screen.



   The Quick Map screen has rows for creating the vertical Edit screen layout for the entity type/map you will define. Select bytes are provided to simplify adding, moving, or copying information you add to this screen.

   Note: Choose PROCESS, NEWMAP to access the ID prompt and reselect a new map ID without exiting the Quick Map facility. When changing a map, enter the map ID of an existing map.

Use the Quick Map Facility

The following sections provide detailed instructions about how to:

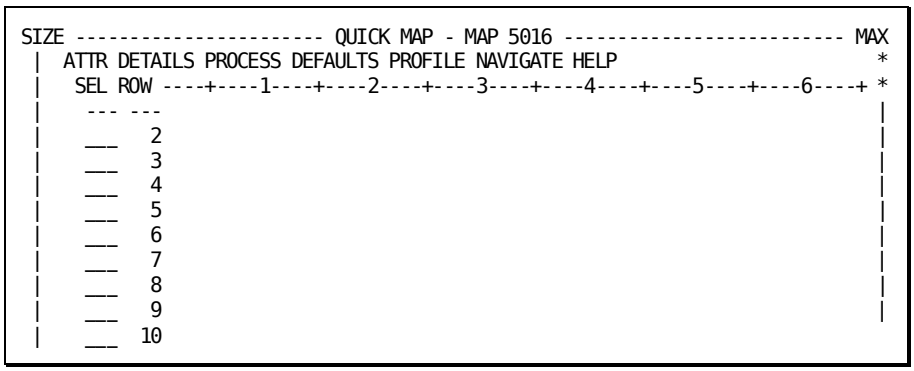
- Add screen literals
- Add input fields
- Customize characters used for input fields
- Define attribute details
- Create on-screen links
- Specify storage tables for maps
- Link the map to an entity type and dialog
- Calculate map offsets
- Recreate an existing DB2 storage table
- Save maps

Add Screen Literals

A screen literal is a block of text or symbols that identifies the purpose and position of an input field.

To add a screen literal

1. Open the Quick Map screen. See Open the Quick Map Facility section earlier in this chapter for more information.



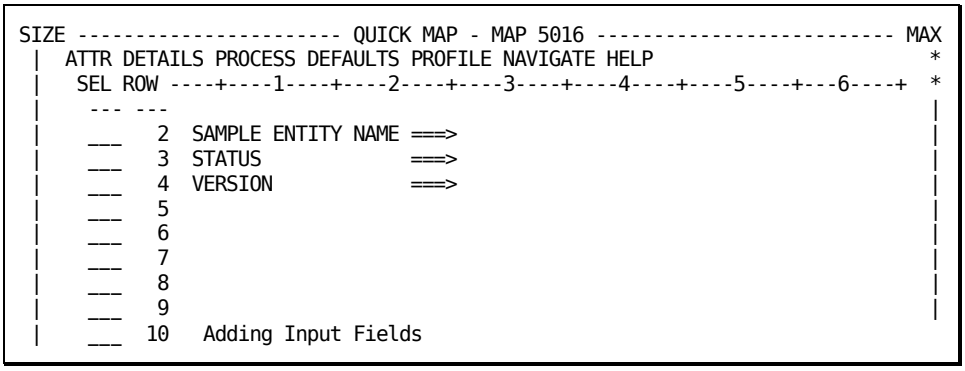
2. Move the cursor to the position on the Quick Map window where you want it to appear and type in the screen literal.

Example of Adding Screen Literals

In the figure below, the following screen literals have been added:

- SAMPLE ENTITY NAME
- STATUS
- VERSION

The SAMPLE ENTITY NAME screen literal identifies the name attribute's input field, while the STATUS and VERSION literals identify the status and version input fields.



An input field is an area on the screen where you can type information for processing. You add input fields using Quick Map the same way you add screen literals, by typing them where you want them.

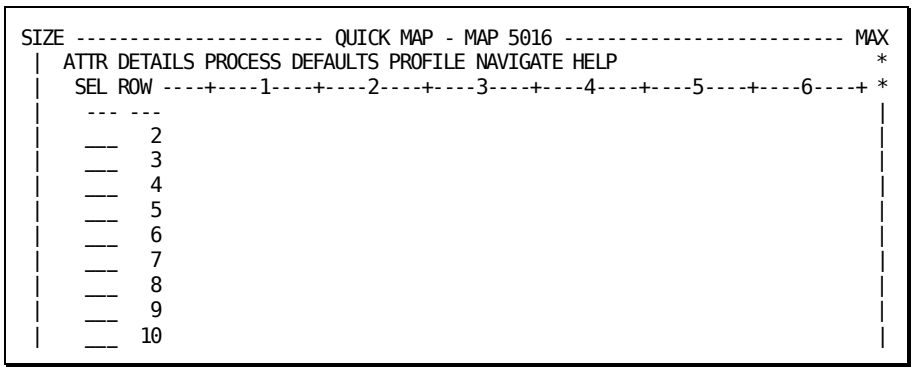
The NAME attribute on CA Repository for z/OS r7.2 maps may be defined as either CHAR or VARCHAR and can have a length of up to 245 characters. The "Name" column, in the underlying r7.2 tables, is always defined as VARCHAR(245).

Additionally, the VERSION attribute on CA Repository for z/OS r7.2 maps must be defined as CHAR(6) to allow the value to be properly displayed. The value is converted to a SMALLINT when stored in the underlying DB2 tables. The IO module also validates that the indicated value is between 0 and +32767 before updating the effected table.

Use the following procedure to add an input field.

To add an input field

1. Open the Quick Map screen.

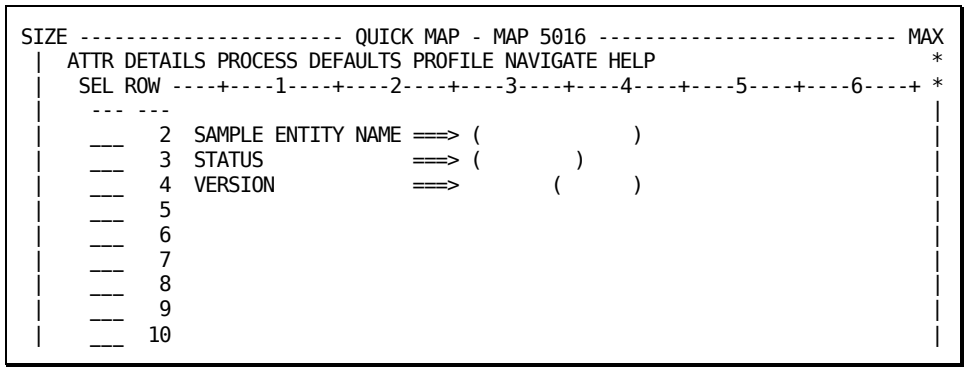


2. Supply your input field information by using the { field marker to mark the beginning of the input field, and the } field marker to mark the end.

   For example, if you want a 10 byte field that starts at column 40, place a begin field marker at column 39 and an end field marker at column 50.

Add Input Fields Example

The following screen shows examples of input fields that were added to the sample map.



Special Formatting Characters

Quick Map provides two additional characters for defining your screen:

- The Hilite literal (!)

  Use the Hilite literal to turn on highlighting for a screen literal or an input field. Highlighting is turned off when another special character is encountered.

- The Lolite literal (@)

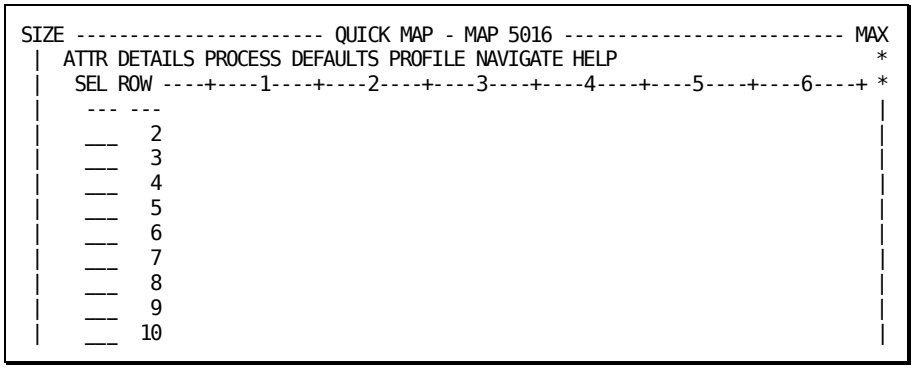
  Use the Lolite literal to turn off highlighting for a screen literal or input field.

Customize Characters Used for Input Fields

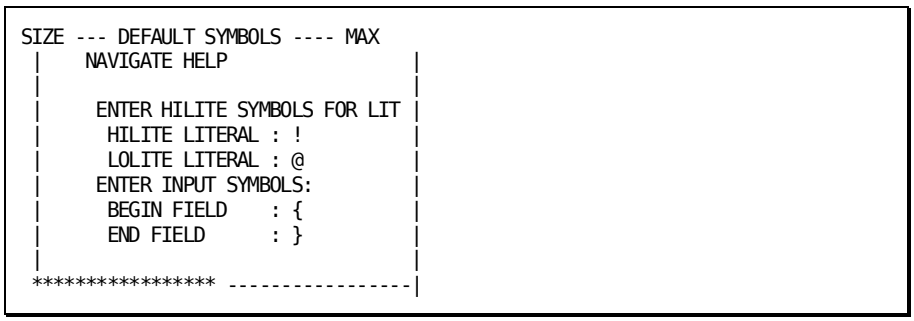
Use the following procedure to customize characters used for input fields (and highlighting literals).

To customize characters used for input fields and highlighting

1. Open the Quick Map screen.



2. Execute the DEFAULTS.SYMBOLS command; this displays the Default Symbols dialog.



3. Select a character you want to change and replace the default character with a character of your choice.

   Note: Be sure to use characters that you are not likely to use as part of a screen literal.

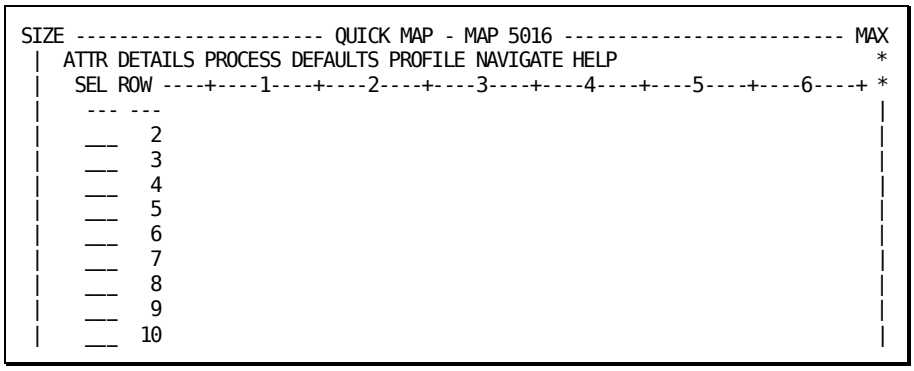
4. Exit the Default Symbols dialog by pressing F3.

Define Attribute Details

When you create new entity types, you must define the actual characteristics of the entity type's attribute types. Use the following procedure to define attribute details.

To define attribute details

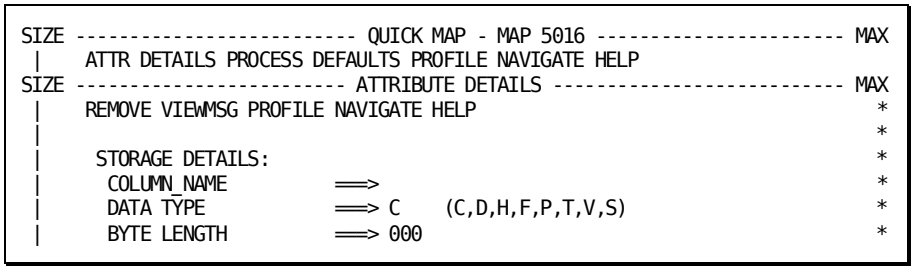
1. Open the Quick Map screen.

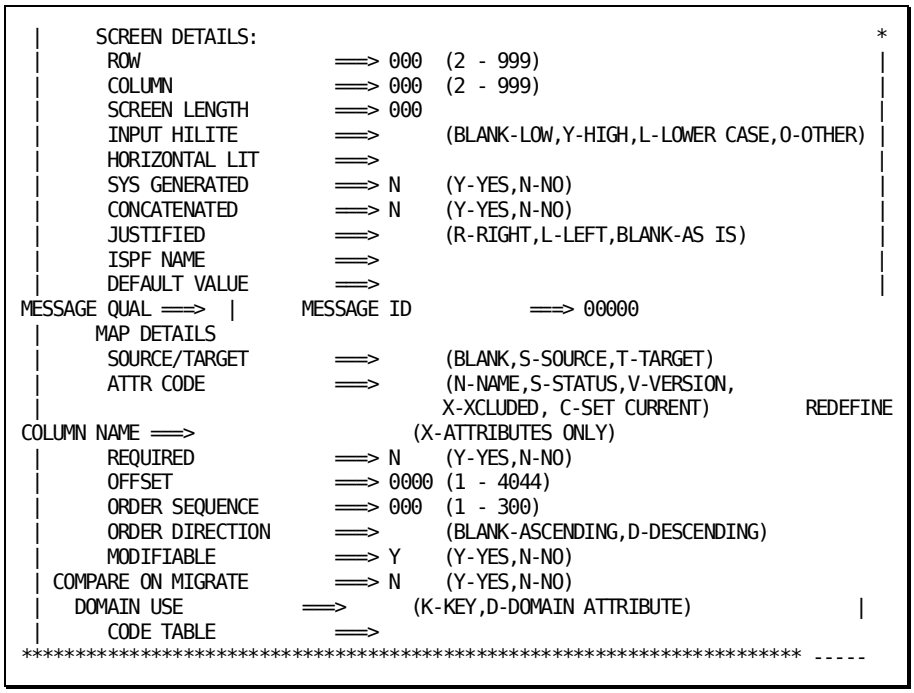


2. Add input fields. See the Add Input Fields Example section, earlier in this chapter.

3. Enter a question mark in one of the input fields on your screen and select the ATTR option from the menu bar to display an Attribute Detail screen, similar to the following illustration.

   Note: Since the attribute type input fields extend beyond one screen, use F7 and F8 to scroll.





4. Enter attribute details as follows:

Field	Enter
COLUMN NAME	The name of the DB2 column in the entity type's underlying storage table used to store this attribute type.
DATA TYPE	Enter one of the following: C-Character D-Date H-Smallint F-Integer P-Decimal T-Time V-VARCHAR S-TIMESTAMP
BYTE LENGTH	The maximum number of bytes used to store attributes of this type **Note:** This value is automatically calculated using the positions of the field markers in the Quick Map screen. Changes made to the information in this field are reflected in the Quick Map screen when you exit Attribute Details screen.
ROW	The row number on the screen that contains the attribute input field. **Note:** Quick Map defaults a value to this field using the current position of the begin field marker. Changes made to the information in this field are reflected on the Quick Map display when you exit the Attribute Details screen.

Field	Enter
COLUMN	The left-most screen column used by the attribute input field. **Note:** Quick Map defaults a value to this field using the current position of the begin field marker. Changes made to the information in this field are reflected on the Quick Map display when you exit the Attribute Details window.
SCREEN LENGTH	The number of bytes available for entering attributes into the input field. **Note:** In most instances, this value is the same as the byte length.
INPUT HILITE	Enter one of the following: Y-highlighted L-lower-case allowed O-other Blank-Non-highlighted
HORIZONTAL LIT	The literal used for this attribute type in the horizontal mode of the Edit window. **Note:** This literal should not exceed the screen length of the attribute type.
SYS GENERATED	Y-If you want the attribute value to be generated by the system. OR N-If you do not want the attribute value generated. **Note:** If set to Y, the input field is protected.
CONCATENATED	Y-If you want the value attribute to be a system-generated concatenation of other attribute values. N-If you do not want the value attribute to be a system generated concatenation of other attribute values. **Note:** If set to Y, the contents of the input filed are derived from the concatenation sequence specified in the Default Values attribute.
JUSTIFIED	R-If you want the value in the input field to be right-justified. L-If you do not want the value in the input field to be right-justified. Leave this field blank if you want to leave the input field as it is.
ISPF NAME	A value used to identify an ISPF variable containing the attribute values. **Note:** If specified, the Repository Editor places the contents of the attribute into the shared variable pool each time it is called. This enables exit procedures to act upon the data before it is passed to the user or changed in the repository.
DEFAULT VALUE	A default value for the attribute and/or the concatenation sequence for fields flagged as concatenated. **Note:** This input field is optional.
MESSAGE QUAL	The three-character message qualifier for the repository help message associated with this attribute type.
MESSAGE ID	The message ID for the repository help message associated with this

Field	Enter
	attribute type.
SOURCE/TARGET	Enter: S-Source T-Target a blank-for neither source or target
ATTR CODE	A flag used to indicate the special usage of the attribute. Valid values are: N-Name S-Status V-Version number X-Excluded attribute C-Set current **Note:** Excluded attributes are those that exist on the screen only and not on the storage table. Excluded attributes are useful when breaking long variables into several input fields for clarity. A "set current" attribute will have its value set to the user's TSO LOGON (CHAR), the system date (DATE), time (TIME), or timestamp (TIMESTAMP) depending on its storage type.
REDEFINE COLUMN NAME	DESCRIPTION **Note:** This attribute is used to tie logical, screen-only attributes (X-attributes) to physical DB2 attributes. A common use for this characteristic is the DESCRIPTION attribute which, though stored in one DB2 column, is tied to two or more fields on the screen display.
REQUIRED	Y-If the attribute is required. N-If the attribute is not required. **Note:** If set to Y, you will not be able to insert entities of this type until a valid value is specified for this attribute.
OFFSET	The physical starting location of the attribute within the I/O template. **Note:** This value must be between 1 and 4044. The offsets should be defined such that the storage areas of different attribute types do not overlap with one another. Use the OFFSET command from the quick Map screen after you finish adding or modifying attributes.
ORDER SEQUENCE	Valid values are: D-Descending order (highest to lowest) A-Ascending order (lowest to highest) **Note:** If left blank, this attribute is not used for sequencing purposes.
ORDER DIRECTION	Valid values are: D-descending order (highest to lowest) A- ascending order (lowest to highest) Leave this field blank if you do not want to use this field for sequencing purposes.
MODIFIABLE	Y-If the attribute value can be changed by subsequent update statements

Field	Enter
	after it has been assigned a value in the repository. N-If the attribute value cannot be changed by subsequent update statements after it has been assigned a value in the repository.
COMPARE ON MIGRATE	Y-If this attribute should be considered significant when comparing entities during migration. N-If this attribute should not be considered significant when comparing entities during migration.
DOMAIN USE	Valid values are: K-To signify that the attribute is the domain key. D-To signify that the attribute is a domain attribute.
CODE TABLE	An identifier for the repository code table containing valid values for this attribute. **Note:** Quick Map will not create code tables. If you need to use one, you need to define it using the Extend model. For more information, see the Creating Code Tables chapter.

5. After you define the characteristics of the selected attribute type, use the end command (press PF3) to return to the Quick Map Window.

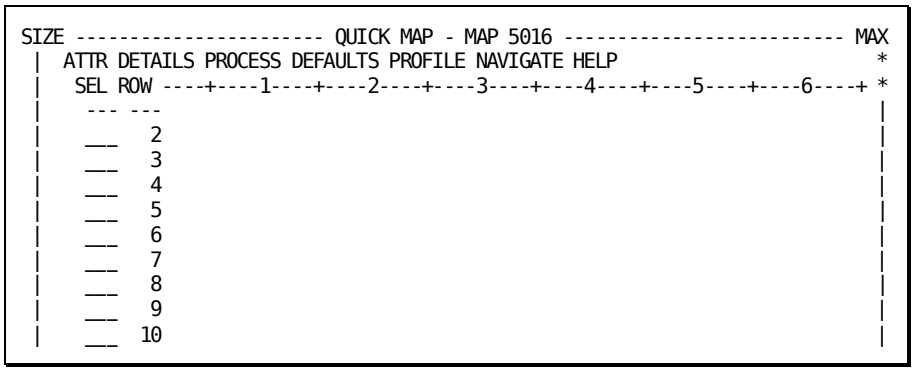
Create On-Screen Links

You can create relationships and associations between entities using maps that do not require you to open a Relationship or Association Edit screen. These maps contain input fields corresponding to two or more entity types. When you insert or update an occurrence using this type of map, the repository automatically creates the relationships or associations between the entities identified by these fields.

To create an on-screen link to another entity type

1. Open the Quick Map screen.

   For information about opening the Quick Map screen, see Opening the Quick Map Facility section, earlier in this chapter.



2. Enter the literal(s) and input field(s).
3. Enter a **?** in the input field and select the ATTR option from the menu bar. The Attribute Detail screen appears. Refer to the illustration in the Defining Attribute Details.
4. Enter values into the following fields:
   - COLUMN NAME
   - SOURCE/TARGET
   - ATTR CODE
   - ON SCREEN TYPE

   Use the following table to determine the values that should be entered:

Field	Enter
COLUMN NAME	The name of the DB2 COLUMN used to store the attribute **Note:** For an on-screen link, this column is on a table other than the one used by the map your defining. For example, if you are creating a map for the TABLE entity type that contains an on-screen link to the TB SPACE entity type, the COLUMN NAME field refers to a COLUMN on the TB SPACE storage table, not the TABLE storage table.
SOURCE/ TARGET	Valid values are: S-If the "linked" entity type is the source of the association/relationship that connects it to the entity type for which you are defining the map. T-If the linked entity type is the target of the association/relationship that connects it to the entity type for which you are defining the map. **Note:** When creating on-screen ties to an entity type connected via a relationship type, you can create fields that reference attributes of the

	relationship itself. In these situations, leave the Source/Target field blank.
ATTR CODE	X-all fields used for on-screen ties.
ON SCREEN TYPE	The ENT TYPE association of the map that ties the two entity types.

5. Add information to the remaining fields of the Attribute Detail as described in the Define Attribute Details section.

   Note: On-screen ties are only possible between entity types that have many-to-one ties between their respective occurrences. Occurrences of an entity type that has on-screen ties to another entity type cannot be related to more than one occurrence of that entity type.

Example of Creating On-Screen Links

The table below shows several of the attribute details for a number of TABLE attributes. Those that are shaded are for on-screen links to the TB SPACE entity type.

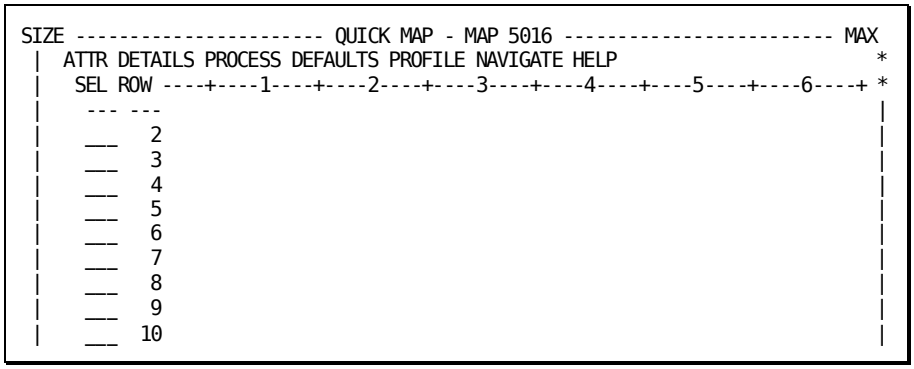
Attribute	Column Name	Data Type	Byte Length	Modifiable	Source / Target	ATTR Code	On Screen Type
TABLE NAME	TB_NAME	C	30	Y		N	
STATUS	STATUS	C	8	Y		S	
VERSION	VERSION	C	6	Y		V	
TS NAME	TABLESPACE_ NAME	C	8	Y	T	X	110
STATUS	STATUS	C	8	Y	T	X	110
VERSION	VERSION	C	6	Y	T	X	110
CREATOR	DB2_CREATOR	C	8	Y			
CREATED BY	CREATED_BY	C	8	N		C	
DATE CREATED	CREATE_DATE	D	10	N		C	
TIME CREATED	CREATE_TIME	T	4	N		C	

Specify Storage Tables for Maps

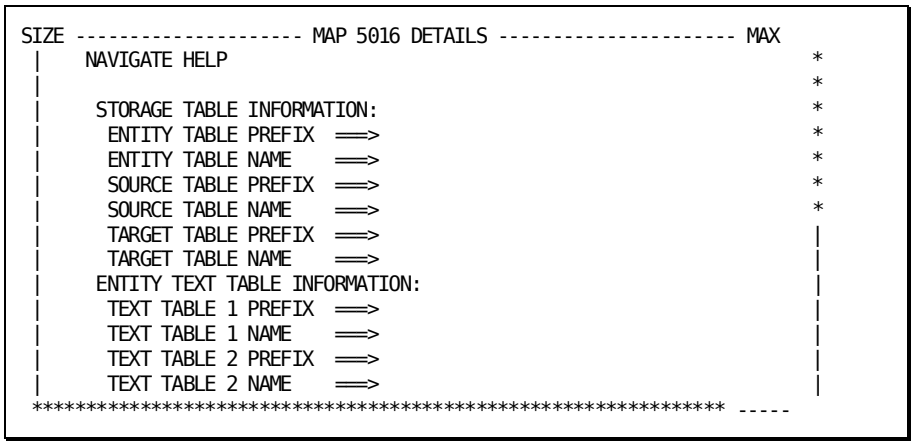
After you define the characteristics for each of the new entity type's attribute types, you must identify the DB2 storage table that will be used to store occurrences of this entity type.

To specify a storage table for a map

1. Open the Quick Map screen. See the section Open the Quick Map Facility earlier in this chapter for more information.



2. Choose DETAILS, MAP; the Map Details screen appears.



Note: Since the input fields of the Map Details screen extend beyond one screen, use F7 and F8 to scroll.

3. Enter values into the Map Details screen as follows:

Field	Enter
ENTITY TABLE PREFIX	The prefix of a table
ENTITY TABLE NAME	The name of the table
TEXT TABLE 1 PREFIX	The prefix of the table that will hold text for this entity
TEXT TABLE 1 NAME	The name of the table 1 name that will hold text for this entity

4. Once you complete the map definition, execute one of the following commands:
   - SYSTEM.EXTEND.CREATE
   - SYSTEM.EXTEND.RECREATE

   Note: There can be up to five text tables for each entity type.

Link the Map to an Entity Type

To link a map to an entity type

1. Open the Quick Map Screen. See the section Open the Quick Map Facility earlier in this chapter for more information.
2. Choose DETAILS, METAENT; the Entity Details screen appears.

```
SIZE ----------------------- ENTITY DETAILS ----------------------- MAX
¦ NAVIGATE HELP R7SP1DEV                                            ¦
¦                                                                   ¦
¦  ENTITY DETAILS:                                                  ¦
¦   ENTITY NAME          ==> ELEMENT                                ¦
¦   ENTITY TYPE          ==> 00204                                  ¦
¦   DESCRIPTION          ==> ELEMENT DEFINITION                     ¦
¦                        ==>                                        ¦
¦   MESSAGE QUALIFIER    ==> REC                                    ¦
¦   MESSAGE ID           ==> 00007                                  ¦
¦  RELATIONSHIP OR ASSOCIATION DETAILS:                             ¦
¦   SOURCE TYPE          ==> 00000                                  ¦
¦   TARGET TYPE          ==> 00000                                  ¦
¦   MAX NUMBER OF SOURCE ==> 00000                                  ¦
¦   MAX NUMBER OF TARGET ==> 00000                                  ¦
¦   ASSOCIATION          ==> N   (Y-YES,N-NO,BLANK-NO)              ¦
¦   DUPLICATES           ==> Y   (Y-YES,BLANK-YES,N-NO)             ¦
¦   HISTORY FLAG         ==> Y   (Y-YES,BLANK-NO,N-NO)              ¦
****************************************************************** -----
```

Note: Since the input fields of the Map Details screen extend beyond one screen, use F7 and F8 to scroll through the input field list.

3. Enter values into the Entity Detail screen input fields as follows:

For	Enter
MESSAGE QUALIFIER	The three character message qualifier that identifies the help message associated with the entity type **Note:** Help messages can be used to create a more detailed description for the entity type than is possible with the Description field.
MESSAGE ID	The message ID for the help message associated with this entity type **Note:** Quick map cannot define help messages for maps and attributes. You will have to define help messages for maps and attributes using the Extend model components.
SOURCE TYPE	The source entity type for relationships and association types. **Note:** If you are creating a relationship or association type, you want to create the source and target types first. If you are defining a standard entity type, you will not need to use this field.
TARGET TYPE	The target entity type for relationship and association types. **Note:** If you are creating a relationship or association type, you want to create the source and target types first. If you are defining a standard entity type, you will not need to use this field.
MAX NUMBER OF SOURCE	The maximum number of times a particular target entity can be used with different source entities when creating relationships or associations with this component.
MAX NUMBER OF TARGET	The maximum number of times a particular source entity can be used with different target entities when creating relationships and associations with this component.
ASSOCIATION	Y-If the map/entity type you are defining is an association type. N-If the map/entity type you are defining is not an association type. **Note:** If the source and target information is specified but this flag is set to N (or left blank), the entity type you are creating will become a relationship type.
DUPLICATES	Y-If you want to allow multiple occurrences with the same source and target occurrences. N-If you do not want to allow multiple occurrences with the same source and target occurrences.

For	Enter
History Flag	Y-If you want to collect history for this object type. N- If you do not want to collect history for this object type.

4. After you have specified all the appropriate entity information, use the END command (press PF3) to return to the Quick Map window.

Link Maps to a Dialog

To link maps to a dialog

1. Open the Quick Map screen. See the section Open the Quick Map Facility earlier in this chapter for more information.
2. Choose DETAILS, DIALOG; this displays an empty Dialog screen.

```
SIZE ---------------------- QUICK MAP - MAP 5016 ----------------------- MAX
 |    ATTR DETAILS PROCESS DEFAULTS PROFILE NAVIGATE HELP                    *
 |   SIZE --------------- DIALOGS --------------- MAX ----+----5----+----6----+ *
 |    |    NAVIGATE HELP
 |    |
 |    |     DIALOGS FOR ENTITY/MAP: 5016
 |    |
 |    |      SEL DIALOG
 |    |      --- --------
 |    |      ___
```

3. Assign the map to an entity type before assigning dialogs.
4. Use line commands to add additional rows to the Dialog screen and enter the appropriate dialog name(s) in the dialog fields.

   Note: If you enter a question mark (?) in the dialog field, a selection list of existing dialogs appears. Any dialog you specify that does not currently exist is added to the control tables as a new dialog.
5. Use the END command (press PF3) to exit the Dialog screen and return to the Quick Map screen.

Note: If the dialog does not yet exist, Quick Map will create it as a new dialog and the ALL DBEXCEL privilege is automatically assigned access to the new dialog.

Calculate Map Offsets

To calculate map offsets

1. Open the Quick Map screen. See the section Open the Quick Map Facility earlier in this chapter for more information.
2. Choose PROCESS, OFFSET. The location of each attribute within the 4044 byte block used by the I/O module to move data to and from the repository will be updated.

   Note: The offset is calculated based on the position and length of other map attributes. Offsets must be calculated each time an attribute within the given map has a change in position or length.

Recreate an Existing DB2 Storage Table

Use the procedure below to recreate an existing DB2 storage table. Adding new attributes to the DB2 storage table will:

- Increase or decrease the size of an attribute
- Change the data type of an attribute
- Delete an attribute from the table
- Change the name of the attribute in a table

To recreate an existing DB2 storage table

1. Open the Quick Map screen. See the section Open the Quick Map Facility earlier in this chapter for more information.
2. Choose PROCESS.RECREATE to build the DDL used to define a new DB2 storage table.

Save Maps

Quick Map provides two options for saving a map, PROCESS, SAVE and PROCESS, SAVENEW:

- PROCESS, SAVE

  This command:

  - Saves a map using the current map ID.
  - Saves updates to maps that you create.

  Use this command to make permanent changes to an existing map. You will not be able save any changes to default maps (1-1000 and >>25,000).

- PROCESS, SAVENEW

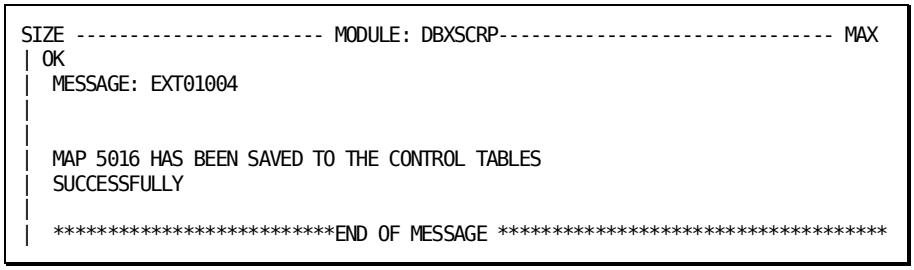
  This command saves a map with a new ID.

  Use this command when making changes to default entity types or when using existing maps as templates for creating new maps.

To save a map

1. Execute either the PROCESS.SAVE or the PROCESS.SAVENEW command directly from the Quick Map window.

   If you use PROCESS, SAVENEW, a window such as the one below appears for each dialog that previously existed.



2. After you save the entity type **/map**, exit Quick Map by pressing **PF3**.

Build History Objects

To build History objects

1. Ensure that the History Flag is set to **Y** for this object.
2. Choose the BLDHST command from the Process window in Quick Map. If the History Flag is set to Y, the History table and its triggers are created. If the History Flag is not set to Y, message EXT01022 appears.

Remove History Objects

To remove History objects

1. Ensure that the History Flag is set to N for this object and all other objects that share the same table.
2. Choose the REMHST command on the Process window in Quick Map. If the history flag is not set to N, message EXT01023 appears. DDL to drop the history table and the history triggers from the metadata tables is generated.

Setting Approval for Objects

To set Object with Approval

1. Ensure that you are chosen correct entity type for Quick Map
2. Set attribute Approval Required on Y

To set Object without Approval

1. Ensure that you are chosen correct entity type for Quick Map
2. Set attribute Approval Required on N or left blank

Chapter 8: Working with Code Tables

This chapter explains how to set up repository code tables to control what value users can specify for certain Repository object attributes.

This section contains the following topics:

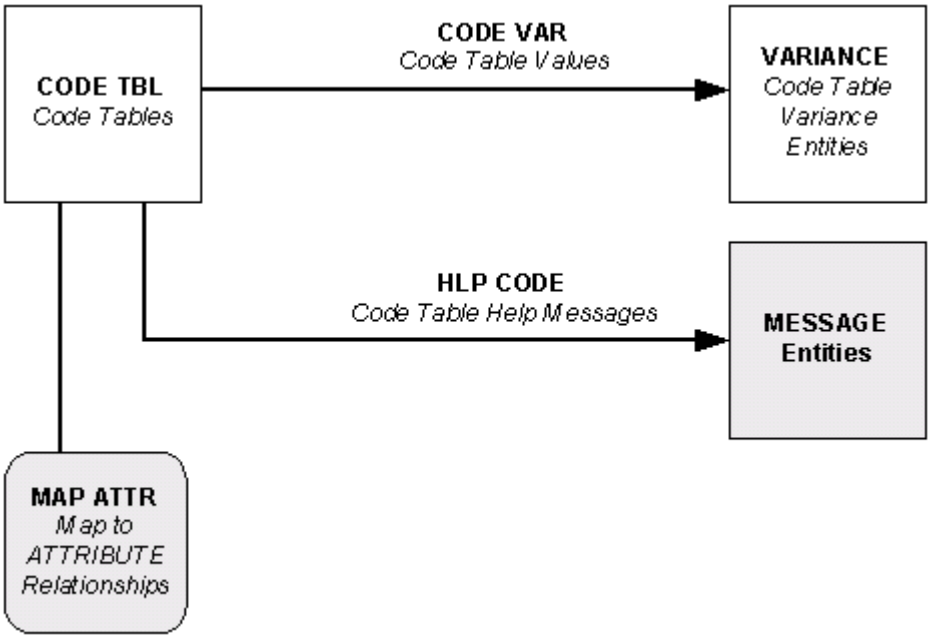
What Is a Code Table? (see page 115)
The Code Table Model (see page 116)
How You Create a Code Table (see page 117)
Modify a Code Table (see page 123)
Extract Code Table Data from the Control Tables (see page 123)
Remove a Code Table (see page 124)

What Is a Code Table?

CA Repository uses code tables to control the values you can use for various repository attribute types. Once defined, a code table not only validates information added to an attribute type, but also provides a selection list of valid values in response to a user inquiry (a question mark in an input field).

The Code Table Model

The following diagram shows the Extend model object types you must use when setting up and maintaining a code table.



Setting up a code table consists of defining one or more instances of each of these object types and then moving the definitions to the repository control tables.

The purpose of each of these object types is as follows:

Object	Function
CODE TBL	Defines individual code tables. All instances that make up a particular code table definition link to a single CODE TBL instance. Building a CODE TBL instance activates the code table in the repository.
CODE VAR	Links a code table definition to one or more VARIANCE instances-sets of allowable values. These associations determine the permissible values for attribute types linked to a particular code table. You can link each CODE TBL entity to up to 300 VARIANCEs through separate CODE VAR associations.
HLP CODE	Links a code table definition to a help message. CA Repository displays a code table message when you

Object	Function
	select HELP from the menu bar of a Code Table window.
VARIANCE	Defines one possible variation (value) that can be assigned to an attribute. Each variance represents a value in several different ways to allow for differences in the types of code tables that can use that particular variance-the code table definition determines which of the variation values CA Repository uses for a particular attribute.

While connected to code table definitions, the MESSAGE and MAP ATTR entity types are not a direct part of a code table definition. Entities of both of these types are typically created as part of other processes and are discussed in detail in the "Customizing Messages" chapter, and the "Adding and Customizing Maps and Entity Types" chapter, respectively.

How You Create a Code Table

All instances that make up a particular code table definition link to a single CODE TBL instance. Building a CODE TBL instance activates the code table in the repository.

Creating a code table consists of the following steps:

1. Insert (define) a CODE TBL instance to define the code table.
2. Define one or more VARIANCE instances assigning values for code tables.
3. Link the CODE TBL instance to one or more VARIANCE instances through CODE VAR associations to specify which values are in the code table.
4. Link the CODE TBL instance to a help message (MESSAGE) to provide on-line help for the code table.
5. Build the CODE TBL instance to the Repository control tables to activate it in CA Repository.

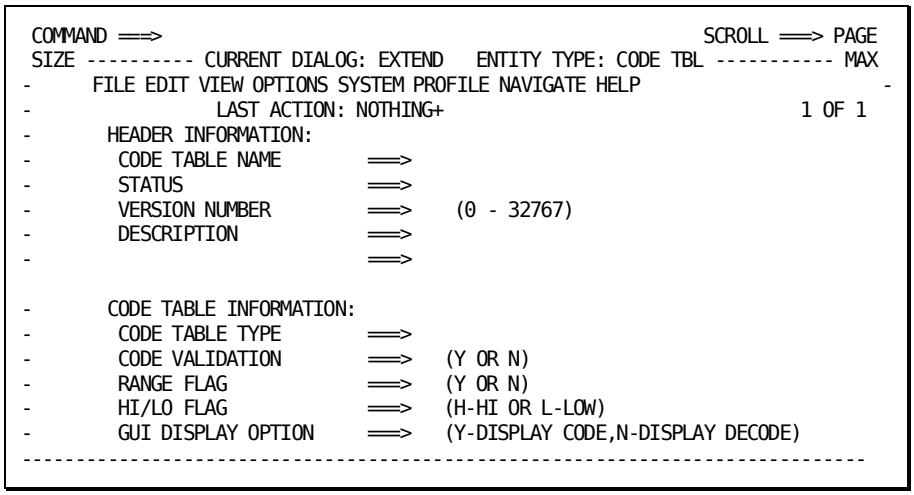
Instructions for each of these tasks are provided in the following sections.

Define a Code Table Instance

The first thing you must to do when creating a new code table is to insert a CODE TBL instance into the repository. This instance acts as the focal point for links to instances of several other object types. This is also the instance you will build (add to the control tables) when you finish inserting all of the necessary object instances.

To add a code table

1. From CA Repository, choose VIEW, DIALOG to access the EXTEND dialog.
2. Choose VIEW, TYPE to access the CODE TBL entity type.



3. Enter values into the CODE TBLE input fields, these fields are as follows:

Field	Enter
Code Table Name	A name for the code table.
Status	The status of the CODE TBL instance.
Version Number	The version number of the CODE TBL instance.
Description	A description of the code table.
Code Table Type	Enter one of the following Code Table Types: 1-Use 1-character variance values. 5-Use 5-character variance values. 10-Use 10-character variance values. DESC-Use the DESC variance value. INTEGER-Use the INTEGER variance value.
Code Validation	Y-Limits user choices to values or ranges of values specified in the associated variance definitions. N-Uses the code table to provide selection lists without validation.
Range Flag	Y-Indicates that the values in the code table define an upper and lower range of acceptable values. N-Indicate that the code table stores specific values.
Hi/Lo Flag	H-Indicates that the value in the code table represents the

Field	Enter
	highest acceptable value in a range of values. L-Indicates that the value in the code table represents the lowest acceptable value in a range of values.
GUI Display Option	Reserved for future use.

4. After entering the information in each of the CODE TBL input fields, choose EDIT, INSERT to insert the code table instance into the repository.

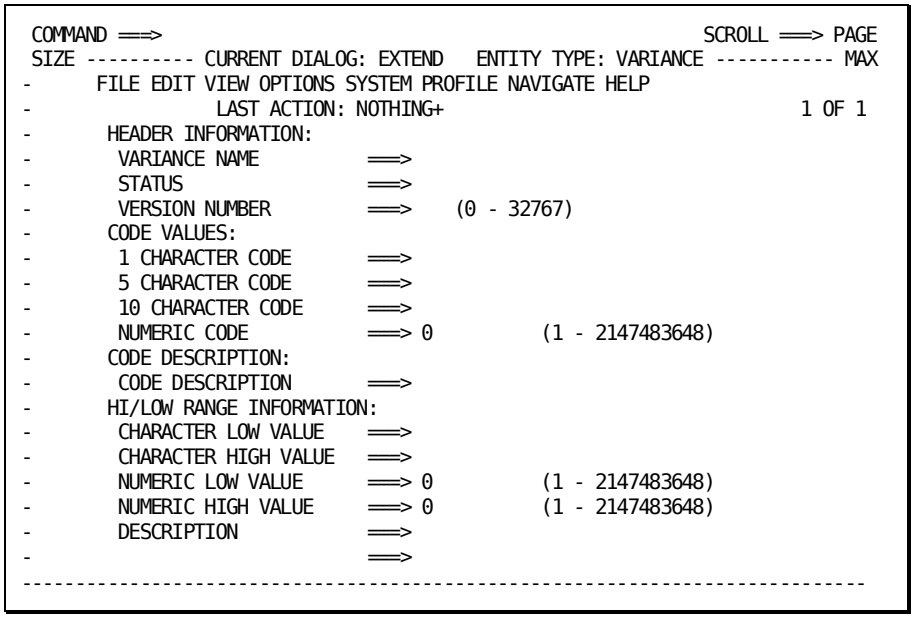
Define a Variance

After you define and insert a code table, you must define the variances to be linked to the code table. Variance instances represent values or ranges of values that you can associate with a particular code table.

For example, if you wanted to create a code table listing states in the USA, you would need to define and insert 50 separate variances, one for each possible value.

To define a variance

1. From CA Repository EXTEND dialog, choose VIEW, TYPE to access the VARIANCE entity type.



2. Enter values in each of the VARIANCE input fields as follows:

Field	Enter
Variance Name	A name for the VARAINCE instance.
Status	The status of the VARIANCE instance.
Version Number	The version number of the VARIANCE instance.
1 Character Code	A 1-character representation of the value defined by this variance.
5 Character Code	A 5-character representation of the value defined by this variance.
10 Character Code	A 10-character representation of the value defined by this variance.
Numeric Code	A numeric representation of the value defined by this variance.
Code Description	A description of the value. This description appears next to the value in the code table. DESC code tables using this variance use this value as the variance.
Character Low Value	The lowest acceptable character value for attributes referencing this variance as a range.
Character High Value	The highest acceptable character value for attributes referencing this variance as a range.
Numeric Low Value	The lowest acceptable integer value for attributes referencing this variance as a range.
Numeric High Value	The highest acceptable integer value for attributes referencing this variance as a range.
Description	A description of the VARIANCE instance.

3. Choose EDIT, INSERT to add the variance to the Repository.

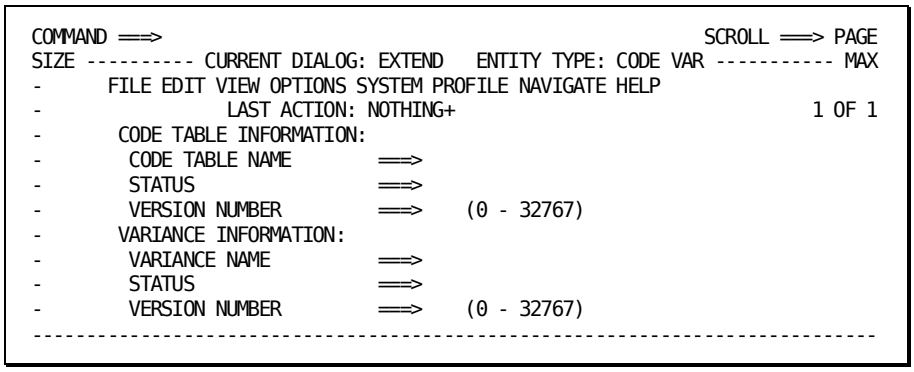
Repeat Steps 2 and 3 for any additional variances that you want to link to the new code table.

Add Variances to a Code Table

Once you define a CODE TBL instance and created one or more VARIANCE instances to represent the values you want in the code table, you need to create links in the repository between the CODE TBL instance and each of the appropriate VARIANCE instances.

To link a code table to a variance

1. Access the CODE VAR association type dialog from the CA Repository Extend dialog by choosing VIEW, TYPE.



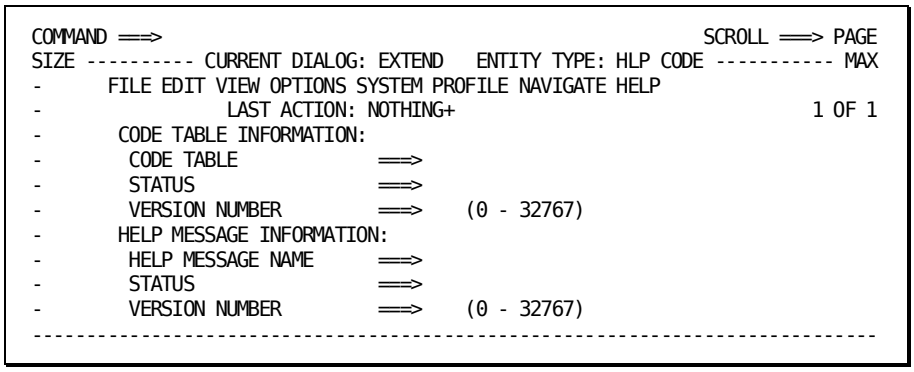
2. Enter the identifiers: name, status, and version in the Code Table Information section of the CODE TBL instance you inserted as part of the Defining a Code Table Instance procedure described earlier in this chapter.
3. In the Variance Information section, enter the identifiers for a VARIANCE instance you inserted as part of the Defining a Variance procedure described earlier in this chapter.
4. Choose EDIT, INSERT to add the association to the repository.
5. If the code table is to have multiple variances, repeat Steps 3 and 4 for any additional variance instances you want to link to the code table.

Provide Help for a Code Table

If you decide you want to provide help for a code table, you must link your CODE TBL instance to a MESSAGE instance. CA Repository displays the message when a user selects help from a Code Table window. Associating a code table with a help message is optional.

To provide help for a code table

1. In the Extend dialog, choose VIEW, TYPE to access the HLP CODE association type.



2. In the Code Table Information section, enter the identifiers (name, status, and version) of the CODE TBL instance you inserted as part of the tasks included in the Define a Code Table Instance procedure described earlier in this chapter.
3. In the Help Message Information section, enter the identifiers (name, status, and version) of the MESSAGE instance defining the help message for this code table. For information on creating CA Repository help messages, see the "Customizing Messages" chapter later in this document.
4. Choose EDIT, INSERT to add the association to the repository.

Add a Code Table to the Control Tables

Once you define a code table and link it to variances and a help message, you need to add it to the control tables in order to activate it within CA Repository.

To add a code table to the control tables

1. In the Extend dialog, choose VIEW, TYPE to access the CODE TBL entity type.
2. Choose VIEW, LIST, ENTITY to select the code table you want to add to the control tables (if you are creating a new code table, this is the CODE TBL instance you inserted in a previous procedure. See Define a Code Table Instance section earlier in this chapter.

3. Choose SYSTEM, EXTEND, BUILD to add the code table and the information contained in its related components to the control tables.

   A message indicating the success or failure of the build appears after processing. If successful, you can now link the code table to one or more map attributes to provide validation and user support.

   Note: Only those components linked to the CODE TBL entity at the time you execute the BUILD will contribute to the operation of the code table. You must define associations linking code tables to variances and messages before you build the code table.

Modify a Code Table

As your requirements change, it may be necessary to modify certain aspects of a code table. Use the following procedure to modify and rebuild a code table.

To modify a code table

1. From CA Repository, choose VIEW, DIALOG to access the Extend dialog.
2. Choose VIEW, TYPE to access the CODE TBL entity type.
3. Choose VIEW, LIST, ENTITY to select the desired CODE TBL instance.
4. Make any necessary changes to the code table definition and use EDIT, UPDATE to save the changes to CODE TBL instance.
5. If necessary, add or remove CODE VAR associations linking the code table definition to VARIANCE instances.
6. If necessary, modify the HLP CODE linking the code table definition to a help message.
7. Reselect the CODE TBL instance and choose SYSTEM, EXTEND, BUILD to replace the CODE TBL entity and all its related components in the control tables with the modified versions. A message indicating the success of the build appears.

Extract Code Table Data from the Control Tables

If the code table data in the Extend model no longer exists or has been altered, you can use the IMPORT command to bring data directly from the control tables to the Extend model storage tables.

WARNING! Be certain that any necessary messages exist in the repository before importing a code table from the control tables.

To import data from control tables to storage tables

1. Choose VIEW, DIALOG to select the Extend dialog.
2. Choose VIEW, TYPE to select the CODE TBL entity type.
3. Choose SYSTEM, EXTEND, IMPORT. A list of code table definitions currently in the control tables appears.
4. Enter **S** in the SEL field of each code table you want to import; then press Enter**.**

   Notes:

   - If the selected code table is not in the Repository model, a progress message appears stating that the import request is being processed. Processing is very rapid, so the progress message may be displayed and removed before you can to read it. Once the process message disappears, a message appears stating that the import has completed.
   - If the code table already exists in the Repository model but was modified, the import progresses as explained previously. However, the system adds a new CODE TBL instance with a higher version number than its pre-existing counterpart.

   You can now use the Editor to select and make modifications to the CODE TBL instance and its associated instances as necessary.

Remove a Code Table

Use the following procedure to remove a code table from the Repository.

To remove a code table from the Repository

1. From CA Repository, choose VIEW, DIALOG to access the Extend dialog.
2. Choose VIEW, TYPE to access the CODE TBL entity type.
3. Choose SYSTEM, EXTEND, REMOVE to display a list of all the code tables currently in the control tables.
4. Enter **S** in the SEL field of each code table you want to remove, then press Enter. A message stating that the remove was successful appears.
5. Use the end command (press PF3) to return to the CODE TBL Edit screen.

   Note: The REMOVE command does not delete or alter any data in the Extend Model.

Chapter 9: Working with Domains

This chapter explains how to create and use domain sets to automate the entry of certain sets of attribute types.

This section contains the following topics:

Domain Keys and Values (see page 125)
Domain Tables (see page 127)
Set Up a Domain Dialog and Domain Tables (see page 128)

Domain Keys and Values

The CA Repository domain feature allows you to link together a set of attribute values in such a way that entering one member of this set into the appropriate Edit window input field automatically enter the remaining values of the set into their respective fields.

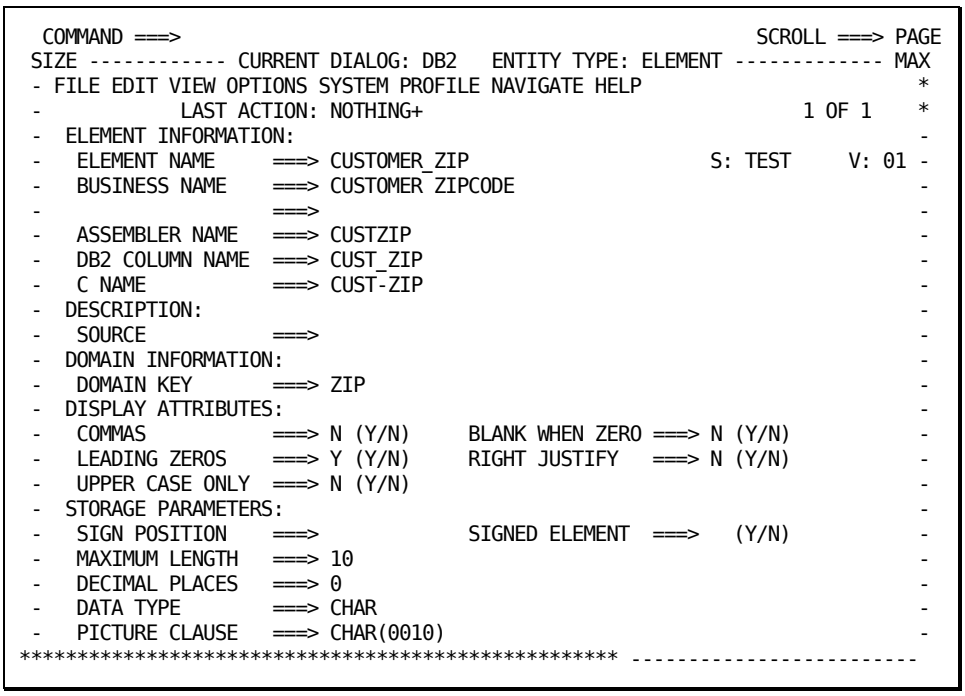
Domain Sets

The attributes types that comprise a domain set can be one of two types:

- **domain keys**-the attribute value a user enters to retrieve the set of domain values.
- **domain values**-the attribute values associated with the domain key the CA Repository retrieves when a user enters a domain key.

Type is based on the attribute type of each value. You specify the type when you define attribute type (MAP ATTR) during the map definition process. If you enter the appropriate value into the input field of an attribute type specified as a domain key, any domain values linked to that key are entered into their respective domain attribute type fields.

The Element Edit Window illustration shows an Edit window for the ELEMENT entity type. The element that is defined in this window is a customer zip code. Since the company defining this element intends to define several types of zip codes and since most zip codes usually have similar characteristics (such as length, and so on), a domain set is specified. Although screen literals for many domain key attributes are usually not this simplistic, they should contain the word *domain*.



Notes:

- If an entity type uses a domain set, one or more attribute types (MAP ATTR) must be labeled as domain keys or domain attributes when defining those attribute types.

  For more information on defining attribute types, see the "Adding and Customizing Maps and Entities" chapter.

- A domain set must have at least one domain key and one domain attribute. In most cases, a single key and several attributes are used.

  If you specify more than one key, a combination of values in all three attribute input fields is required to retrieve the associated domain attribute values.

  You should associate a code table with a domain key attribute to allow users to generate selection lists of the available domain keys.

Domain Tables

A domain table is where the repository stores domain values. When you define maps, you also specify the prefix and name of the domain table.

You can create and use two types of domain tables:

- One table for each entity type using domain sets
- One table for several separate domain sets

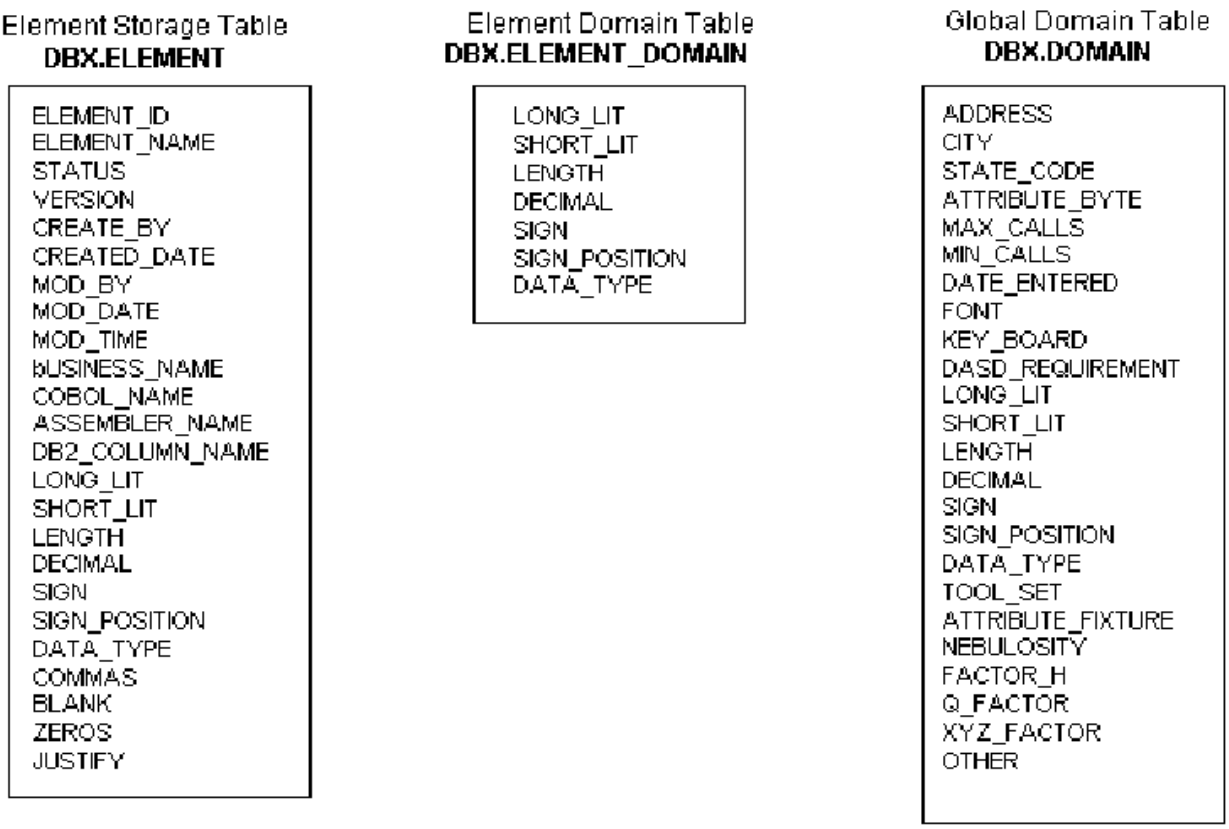
Notes:

- Any map containing map attributes designated as domain keys or domain attributes must also contain the specification of the appropriate domain table.
- Since a domain set is a subset of the attribute types of an entity type, a domain table must contain a subset of the columns in the entity type storage table.

For example, if the ELEMENT entity type consists of 25 attribute types-its storage table contains 25 columns corresponding to those attribute types. Of those 25 attribute types, only 7 are domain key or domain attributes. The domain table for the ELEMENT entity type contains 7 columns analogous to those in the ELEMENT storage table uses to store the domain attribute data.

If you think many entity types at your site will use domain sets, it may be easier to define a single domain table. This table would be composed of all those columns used as part of a domain set throughout your data models. If only a few entity types will use domains, it may be as easier to define a separate domain table for each entity type.

See the Storage Table and Domain Table Combination illustration that follows.



Set Up a Domain Dialog and Domain Tables

To create a domain dialog and a set of entity type-specific domain tables, using the following procedure.

To create a domain dialog and domain tables

1. From CA Repository, choose VIEW, DIALOG to open the EXTEND dialog.
2. Create an insert a DIALOG instance defining a Repository dialog for your set of domain entity types.
3. Using Quick Map (OPTIONS, QUICK, MAP), open the map for the entity type for which you're creating a domain table.

   Note: For the procedure on opening and using Quick Map, see the "Adding and Customizing Maps and Entities" chapter.
4. Edit the map to remove all attributes that are not part of the domain set.

5. Choose DETAILS, MAP to specify a new table name.
6. Choose DETAILS, DIALOG to link the map to the DIALOG instance created in step 2. Be sure to eliminate and links to other dialogs.
7. Choose PROCESS, NEW to save the map with a new ID.
8. Choose the Quick Map's PROCESS, CREATE option to create a storage table for new entity type. Be sure to change the name of the table.
9. Repeat steps 3-8 for any other entity types for which you'd like to create a domain table.

To populate a domain using the domain dialog

1. Choose VIEW, DIALOG to open the dialog containing to the domain-only entity types (Step 2 above).
2. Choose VIEW, TYPE to select an entity type.

   Selecting an entity type in the domain dialog opens an Edit screen for maintaining *domain* data rather than the entity data.

   The following screen shows an Edit screen for a hypothetical domain entity type. This ELMNT_DM entity type contains attributes analogous to those that are part of the domain set for the ELEMENT entity type.

   Each entity defined in this window is added to the domain table as a domain set for the ELEMENT entity type.

```
COMMAND ===>                                                  SCROLL ===> PAGE
SIZE ---------- CURRENT DIALOG: DOMAIN   ENTITY TYPE: ELMNT_DM ----------- MAX
- FILE EDIT VIEW OPTIONS SYSTEM PROFILE NAVIGATE HELP                        -
-            LAST ACTION: NOTHING+                              1 OF 1     -
-  DOMAIN INFORMATION:                                                      -
-   DOMAIN KEY        ===> ZIP                                              -
-   STATUS            ===> DOMAIN                                           -
-   VERSION NUMBER    ===> 01 (0 - 32767)                                 -
-                                                                           -
-   MAXIMUM LENGH     ===> 10                                               -
-   DECIMAL PLACES    ===> 0                                                -
-   SIGNED ELEMENT    ===> N (Y OR N)                                       -
-   SIGN POSITION     ===>   (L-LEADING,T-TRAILING)                         -
-   DATA TYPE         ===>   (Y-DISPLAY CODE,N-DISPLAY DECODE)              -
-   PICTURE CLAUSE    ===> CHAR(0010)                                       -
-------------------------------------------------------------------------------
```

3. Insert an instance for each set of domain attributes. Be sure the domain key attributes are different for each instance you insert.

Chapter 10: Customizing Messages

This chapter explains how to customize the LOGON, LOGOUT, help, and supplementary messages.

This section contains the following topics:

The CA Repository Messages (see page 131)
The Message Model (see page 133)
Create a Help Message (see page 133)
Customize Existing Messages (see page 138)
Remove a Message (see page 142)

The CA Repository Messages

Messages do the following:

- Support communication between the system and the user.
- Provide information in the form of help messages or as warnings.

Since each DB2 environment has its own specific requirements, default messages may not always be appropriate for every application or situation. You can modify messages with the proper authorization. If necessary, you can create entirely new messages.

Types of CA Repository Messages

The messages you can encounter in CA Repository are classified into the following types:

- Information messages
- Progress messages
- Warning messages
- Action messages
- ISPF messages and tutorial panels

Since each of these message types provide you with a different kind of information, each will prompt you for a different type of response.

Message Type	Purpose and Description
Information	■ Informs you that a process has been executed normally

Message Type	Purpose and Description
	■ Provides additional information on a specific topic or field in direct response to a user request Information messages are the most common type of CA Repository messages
Progress	■ Lets you know the status of a particular process ■ Informs you that the desired action is progressing normally ■ Contains no icons to solicit a response. **Note:** The display of progress messages may be so quick that they offer no useful information. During lengthy processes, these messages assist you in understanding the status of the current process.
Warning	■ Informs you that a potentially undesirable result may occur (if the present process continues) ■ Provides you with the option of either continuing or terminating processing ■ Prompts you with Yes and No icons (Warning 1 messages) or OK and CANCEL icons (Warning 2).
Action	■ Informs you of the occurrence of an error or exception that requires corrective action. ■ Prompts you to either Retry or Cancel the operation.
ISPF Messages and Tutorial Panels	If an ISPF message is defined to the repository, the SETMSG command executes so the next panel display shows the desired message. ISPF Tutorials are invoked by help messages if a panel ID for the tutorial is specified. In such instances, dialog control is passed from the repository to ISPF.

The Message Model

The Message model shown below contains the entity, relationship, and association types of the Extend model that affect-or are affected by-messages. Only MESSAGE and HLP MSG are a direct part of message administration. The remaining components can be linked to Help messages.

The components of the Message model are explained in the table below.

Object	Purpose
MESSAGE	Defines CA Repository messages. Instances of this entity type have attributes corresponding to message characteristics (such as message type, message window position, text width, and so on). The messages that appear to users are extended text of the MESSAGE entities.
HLP MSG	Links one help message to another. This allows you to set up a series of sequential help messages.
HLP ATTR	Links a message definition to a attribute definition to define help for an attribute type.
HLP CMD	Links a message definition to a command definition to provide help for a command.
HLP CODE	Links a message definition to a code table definition to provide help for a code table.
HELP ENT	Links a message definition to a entity type definition to provide help for an entity type.
HLP MIG	Links a help message definition to a migration path to provide help for a migration path.

Create a Help Message

To add a help message

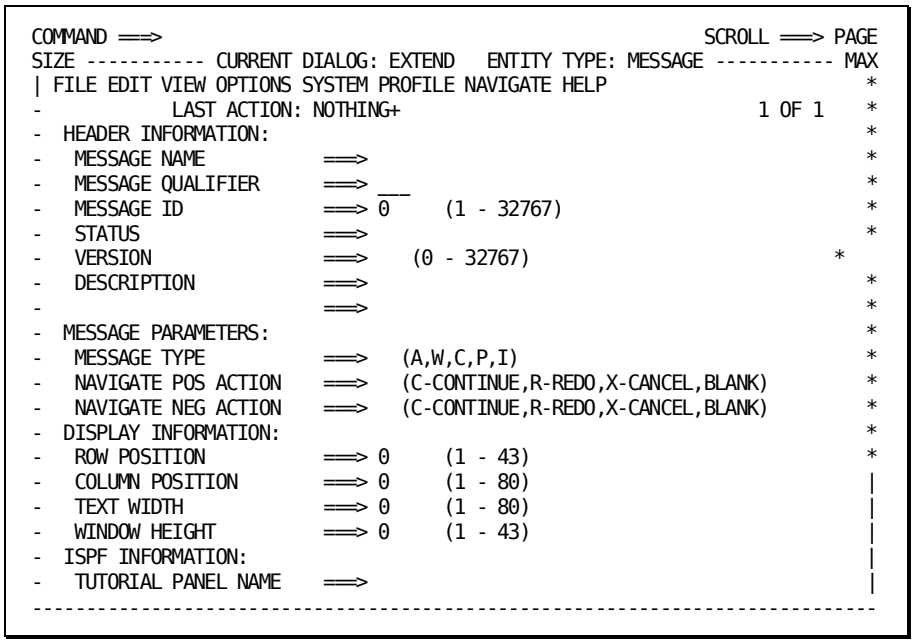
1. Insert a MESSAGE instance, including extended text defining the actual message.
2. Optionally, you can insert HLP MSG associations linking the new help message to any supplement messages
3. Use the SYSTEM.EXTEND.BUILD command to activate the message, adding the MESSAGE instance to the CA Repository control tables.

Insert a MESSAGE Instance

When creating a new CA Repository message, you must first create a MESSAGE instance defining the message. Each MESSAGE instance not only defines the message type (Warning, Informational, Progress, Action), but it also contains the text as seen by CA Repository users.

To add a help message

1. From CA Repository, choose VIEW, DIALOG to access the Extend dialog.
2. Choose VIEW, TYPE to access the MESSAGE entity type.



3. Enter values into the MESSAGE input fields as follows:

For	Enter
Message Qualifier	A qualifier for the message The qualifier allows you to group messages by product type (for example, "DB2").
Message ID	A 5-digit ID uniquely identifying the message instance for a particular qualifier
Status	The status of the MESSAGE entity
Version	The version of the MESSAGE entity
Description	A description of the MESSAGE entity
Message Type	Enter one of the following:

For	Enter
	I-An Information message W-A warning 1 message C-A Warning 2 message A-An Action message P-A Progress message
Navigate Pos Action	The action taken in a Navigation when a message with a positive response (OK or Yes) appears
Navigate Neg Action	The action taken in a Navigation when a message with a negative response (No or Cancel) appears
Row Position	The row position of the message window
Column Position	The column position of the message window
Text Width	The text width of the message
Window Height	The number of rows to display in the message
Tutorial Panel Name	The help panel name
Message Name	The ISPF message identifier for a message in the ISPLIB allocation

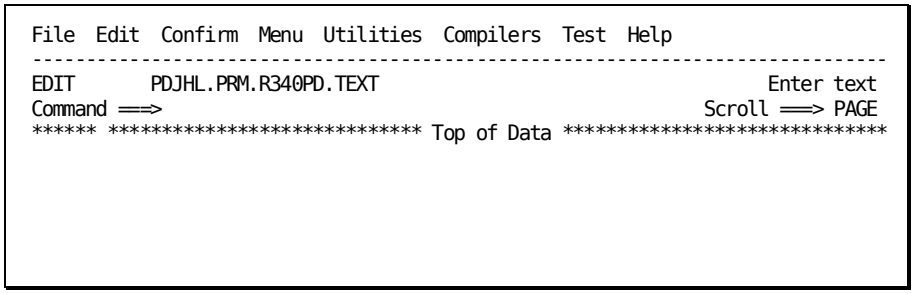
4. Use the EDIT, INSERT command to add the entity to the repository.

Add Message Text to a Message

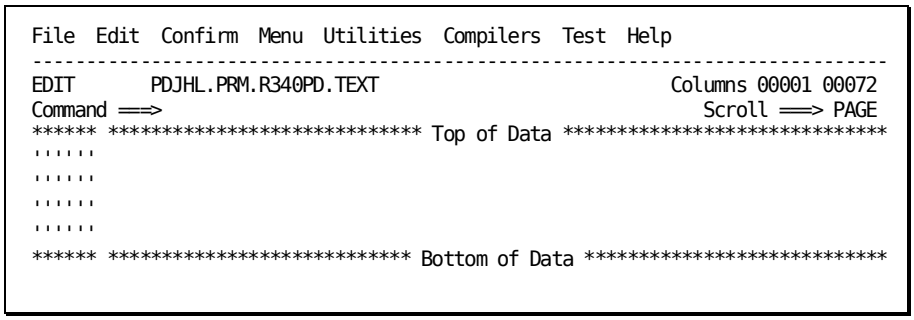
After creating a MESSAGE instance, you'll need to add the extending text defining the message.

To add message text to a message

1. With the appropriate MESSAGE instance displayed, choose EDIT.TEXT.DESCRIPT to open the ISPF Edit Screen for adding extended text.



2. Use the ISPF line command I to add text rows. For example, I4 adds four rows as shown next.



3. Enter your message in the new rows. You can use the following tags:

Tag	Description
=V#	Variable Substitution The repository allows up to ten variables (0-9) to be inserted into a message from the REASON variable. Each variable is separated by one byte containing high values. The message text contains the number of the variable to be substituted from the Reason variable. With the exception of these tags, all message text is treated as text data.
=HI	Highlight (bold) the text that follows
=LO	Lowlight-Marks the end of highlighted text
//	Line Break-Start a new line after this tag

Important! A colon (:) cannot be used in message text as the first character in a word. Colons used at the beginning of words cause a DB2 error each time the message is processed.

Each message can contain up to 4000 bytes of text (including 80 bytes of variables). Although text is stored in 80 byte rows, how it displays depends on both the message type and the width attribute specified in the message definition. The line commands referenced in the Editor manual are available to aid in the manipulation of the message text.

4. Use the END command (press PF3) to exit the text panel and save the message text.

Add Supplemental Messages

You can add supplemental messages that are accessed directly from a help message window by defining and inserting a HLP MSG association. The supplemental message is displayed when a user selects the HELP option from the action bar of a Help window.

To add supplemental messages

1. In he Extend dialog, choose VIEW, TYPE to access the HLP MSG association type. The HLP Edit Window is displayed.

```
COMMAND ===>                                                  SCROLL ===> PAGE
SIZE ----------- CURRENT DIALOG: EXTEND   ENTITY TYPE: HLP MSG ----------- MAX
  FILE EDIT VIEW OPTIONS SYSTEM PROFILE NAVIGATE HELP                        -
-                LAST ACTION: NOTHING+                                  1 OF 1
-       MESSAGE INFORMATION:
-        MESSAGE NAME           ===>
-        STATUS                 ===>
-        VERSION NUMBER         ===>    (0 - 32767)
-       HELP INFORMATION:
-        HELP MESSAGE NAME      ===>
-        STATUS                 ===>
-        VERSION NUMBER         ===>    (0 - 32767)
------------------------------------------------------------------------------
```

2. In the MESSAGE information section, specify the name, status, and version of the help message for which you'd like to provide supplemental help.
3. In the HELP information section, specify the name, status, and version of the instance defining the supplemental help message.
4. Choose EDIT.INSERT to insert the supplemental message into the repository.

   Note: The target message of these associations should contain text that augments or clarifies the text of the source entity. Each message can reference only one other through these associations.

Add a Message to the Control Tables

Once you have defined a message and made necessary associations to other supplemental messages, you'll need to add the message to the control tables in order to make it accessible to CA Repository users.

WARNING! If your help message references a supplemental message through a HLP MSG association, be sure to add the supplemental message to the control tables before you add the message that references it.

To add a message to the control tables

1. From the CA Repository, choose VIEW, DIALOG to access the Extend dialog.
2. From the Extend dialog, choose VIEW.TYPE to access the MESSAGE entity type.
3. Use EDIT.SELECT or VIEW.LIST.ENTTY to retrieve the message instance you want to add to the control tables.
4. Choose SYSTEM, EXTEND, BUILD to build the message.

   The BUILD command adds the message and any HLP MSG associations to the control tables. A message indicating the success or failure of the build appears after processing.

Customize Existing Messages

As your site's requirements change, it may be necessary to modify a message. Use the following procedure to modify and rebuild a message.

To modify a message

1. From CA Repository, choose VIEW, DIALOG to access the Extend dialog.
2. Choose VIEW, TYPE to access the MESSAGE entity type.
3. Use EDIT, SELECT or VIEW, LIST, ENTITY to select the desired MESSAGE instance.
4. Make any necessary changes to the MESSAGE definition and use EDIT, UPDATE to save the changes.
5. If you need to edit the text of the message, choose EDIT, TEXT, DESESCRIPT to display and edit the message text.
6. If necessary, add or remove HLP MSG associations linking the message definition to another message instance.
7. Reselect the MESSAGE instance and use the SYSTEM.EXTEND.BUILD command to replace the MESSAGE definition and any related components in the control tables with the modified versions. A message indicating the success of the build appears.

Extract Code Table Data from the Control Tables

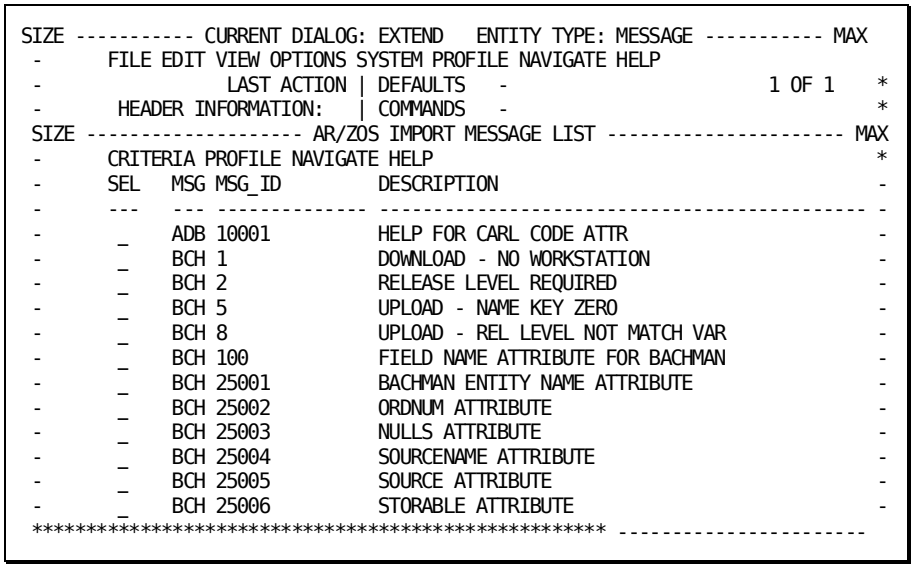
If the message data in the Extend model no longer exists, or if you suspect that it has been altered, you can import message data from the control tables to the Extend model tables, thereby recreating the MESSAGE entity exactly as it currently exists in the control tables.

WARNING! Make certain that any necessary objects instances exist in the repository before importing a message from the control tables.

To import data from control tables to storage tables

1. From CA Repository, choose VIEW, DIALOG to select the Extend dialog.
2. Choose VIEW, TYPE to select the MESSAGE entity type.

3. Choose SYSTEM, EXTEND, IMPORT to display a list of message definitions currently in the control tables.



4. Enter **S** in the SEL field of each message you want to import, and then press Enter.

   Notes:

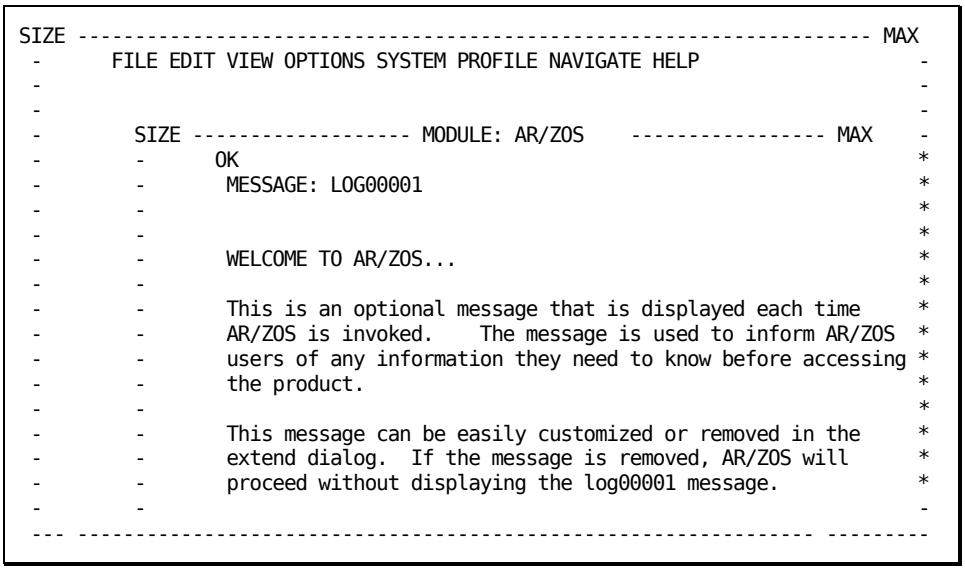
   - If the selected message is not in the Extend model, a progress message appears stating that the import request is being processed. Processing is very rapid, so the progress message may be displayed and removed before you can to read it. Once the process message disappears, a message is displayed stating that the import has been completed.
   - If the message already exists in the repository model but has been modified, the import progresses as explained above. However, CA Repository adds a new MESSAGE instance with a higher version number than its pre-existing counterpart.

5. Use the Editor to select and make modifications to the CODE TBL instance and its associated instances as necessary.

Customize the LOGON Message

The LOGON Message feature supports communication between the administrator and the users accessing the repository. The LOGON message is displayed each time a user logs on to the repository. Use the LOGON message for notifications of system changes, pending maintenance, or other downtime.

Note: The LOGOUT message, LOG00002 can also be customized using the same procedure shown on the following page.



To customize the LOGON message

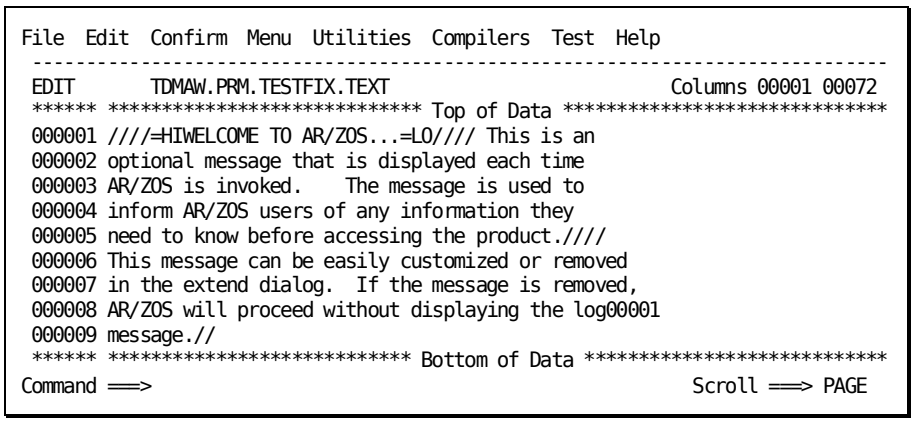
1. From CA Repository, choose VIEW, DIALOG to access the Extend dialog.
2. Choose VIEW, TYPE to access the MESSAGE entity type; the Edit screen is displayed.
3. Enter values into the fields as follows:

Field	Enter
Message Qualifier	LOG
Message ID	1
Status	EXTP
Version	00

4. Choose EDIT, SELECT; the Edit screen for the LOG00001 entity instance appears.

   Note: If the message is not found, use SYSTEM.EXTEND.IMPORT to copy it from the control tables to the Extend model.

5. Once you've retrieved the LOG00001 MESSAGE instance, choose EDIT.TEXT.DESCRIPT to display the message text. The LOGON message text appears similar to that shown in the figure below.



6. Modify the extended text as necessary.
7. Use PF3 to exit the text editing session. CA Repository automatically saves any changes to the text that you've made.
8. Use SYSTEM.EXTEND.BUILD to add the changes to the control tables.

Remove a Message

To remove a message from the repository control tables

1. From CA Repository, choose VIEW.DIALOG to access the Extend dialog.
2. Choose VIEW, TYPE to access the MESSAGE entity type.
3. Use SYSTEM.EXTEND.REMOVE to display a list of all message entities in the control tables.
4. Enter S in the SEL field of the message entity you want to remove, and then press Enter.

WARNING! Selecting a message from the Remove screen removes that message from the control tables and eliminates any associations of the message to other entities. The REMOVE command does not affect the data in the Extend model.

Chapter 11: Working with Keywords

This chapter explains how to add, modify, remove, and extend product keywords to specific maps.

This section contains the following topics:

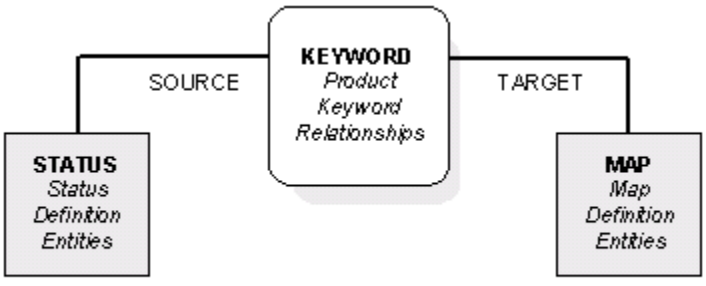
Keywords (see page 143)
Add a Keyword (see page 144)
Modify a Keyword (see page 145)
Remove a Keyword (see page 147)
Extend Keyword Reference Tables (see page 147)

Keywords

In the CA Repository, keywords control which object type maps a repository product uses to store and retrieve repository data.

The repository provides a standard list of keywords. Although you cannot add to the keyword list, you can link the list to an infinite number of usages (usage determines what maps are used) and tailor them to almost any site-specific requirement.

The Extend model object types used to set up and configure keywords are shown in the following illustration. Only KEYWORD relationships are defined as part of keyword administration. STATUS and MAP entities are defined as part of other processes and are explained in the "Working with Statuses" chapter and the "Adding and Customizing Map and Entity Types" chapter, respectively.



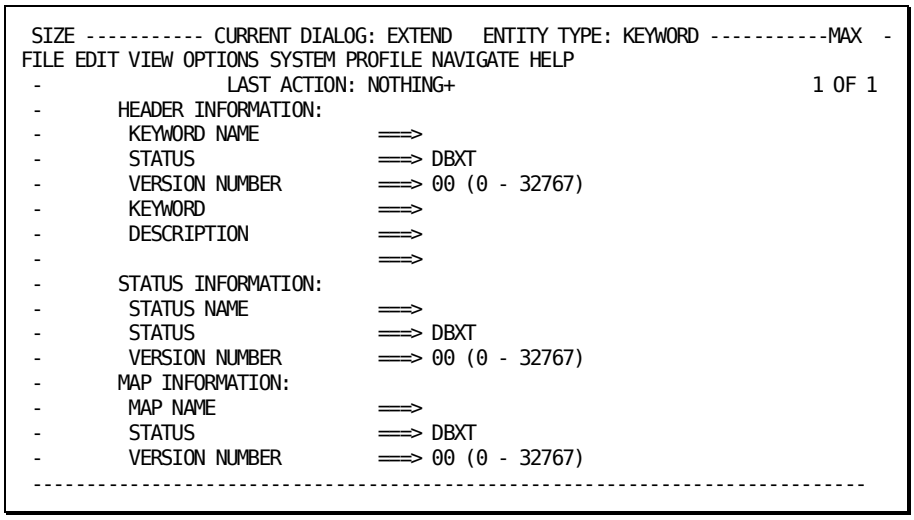
Keywords Model Components

The KEYWORD relationship type is used to determine the map that a product function uses when processing a user directive. Each keyword is stored in the repository as a KEYWORD relationship. These relationships reference both a status/usage (source) and a map (target). When a user performs a repository function that requires a keyword, the Repository prompts the user for a usage. Specify the repository usage that is used by CA Repository to select maps for processing.

Add a Keyword

To add a keyword to the Repository and the control tables

1. From CA Repository, choose VIEW.DIALOG to access the EXTEND dialog.
2. Choose VIEW.TYPE to access the KEYWORD entity type; the KEYWORD Edit window is displayed.



3. Enter values into the KEYWORD fields as follows: .

Field	Enter
Keyword Name	Nothing. This field is system-generated; leave it blank.
Status	A status for the KEYWORD relationship
Version Number	A version for the KEYWORD relationship
Keyword	A descriptive name used to select a CA Repository map in combination with a CA Repository usage
Description	A description of the keyword

4. In the Status Information section, specify the name, status, and version of the STATUS instance that defines the repository status that you would like to have reference the target map through this keyword relationship.
5. In the Map Information section, specify the name, status, and version of the MAP instance you'd like to tie to the status specified in Step 4.
6. When you've completed each of the KEYWORD fields, choose EDIT, INSERT to save the KEYWORD instance to the Extend model tables.
7. Use SYSTEM.EXTEND.BUILD to add the keyword to the control tables.

Modify a Keyword

As your site's requirements change, it may become necessary to modify a keyword definition.

To modify a keyword

1. From CA Repository, choose VIEW, DIALOG to access the Extend dialog.
2. Choose VIEW, TYPE to access the KEYWORD entity type.
3. Use EDIT, SELECT or VIEW, LIST, ENTITY to retrieve the instance defining the keyword you want to modify.
4. Make changes to the KEYWORD definition as necessary and choose EDIT.UPDATE to save the changes to the Extend model.
5. Choose SYSTEM.EXTEND.BUILD to replace the keyword information in the control tables with the modified version.

Import Data from Control Tables to Storage Tables

If the keyword data in the extend model no longer exists, or if you suspect that it has been altered, you can bring data back from the control tables to the storage tables and recreate the KEYWORD relationship exactly as it exists in the control tables.

To import data from the control tables to the Extend model

1. From CA Repository, choose VIEW.DIALOG to access the EXTEND dialog.
2. Choose VIEW, TYPE to access the KEYWORD entity type.

3. Execute the SYSTEM, EXTEND, IMPORT command. A list of the keyword relationships currently in the control tables is displayed.

```
SIZE ----------- CURRENT DIALOG: EXTEND   ENTITY TYPE: KEYWORD ----------- MAX
-      FILE EDIT VIEW OPTIONS SYSTEM PROFILE NAVIGATE HELP                  -
-                 LAST ACTION | DEFAULTS   -                          1 OF 1 -
-       HEADER INFORMATION:   | COMMANDS   -
-        SIZE ------ AR/ZOS IMPORT KEYWORD LIST ------ MAX
-         -      CRITERIA PROFILE NAVIGATE HELP       *                       -
-         -      SEL   KEYWORD  USAGE    MAP_ID         ENT_T  -              -
-         -      ---   -------- -------- -------------- -----  -              -
-         -        _    ADDB LOC ADABAS   26405          26305  -              -
-         -        _    ADDB SEC ADABAS   26406          26306  -              -
-         -        _    ADDBCKPT ADABAS   26407          26307  -              -
-         -        _    ADDBKYWD ADABAS   26404          26304  -              -
-         -        _    AL FOR   ADABAS   601            101    -              -
-         -        _    AL FOR   CARLETON 601            101    -              -
-         -        _    AL FOR   DB2      601            101    -              -
-         -        _    AL FOR   INFORMIX 601            101    -              -
-         -        _    AL FOR   ORACLE   601            101    -              -
--- ------------------------------------------------ ------------------------
```

Note: When using IMPORT, only KEYWORD relationships are returned to the Extend model tables. If the appropriate MAP or STATUS entity is no longer in the Extend storage tables, the keyword relationship cannot exist and the import will fail. To avoid this problem, ensure that the necessary MAP and STATUS entities exist in the repository before importing a keyword. Valid statuses for extensions are EXTP, EXTS, and EXTT.

4. Enter **S** in the SEL field of the keyword(s) you want to import, and press enter. A progress message appears stating that the import request is being processed. Processing is sometimes very rapid, so the message may be displayed and removed before you are able to read it.

   Be aware of the following:

   - If the selected keyword is not in the Repository model, the import proceeds normally. When processing completes, a message stating that the import is complete appears.
   - If the keyword already exists in the Repository model but has been modified, the import proceeds as explained above. However, the system adds a new Keyword entity with a version number greater than that of its pre-existing counterpart.

5. Make changes to the keyword definition as required.
6. Use SYSTEM, EXTEND, BUILD to return the altered keyword to the control tables.

Remove a Keyword

If you must disable a keyword, simply remove it from the repository control tables.

To remove a keyword from the control tables

1. From CA Repository, choose VIEW.DIALOG to access the EXTEND dialog.
2. Choose VIEW.TYPE to access the KEYWORD entity type.
3. Execute the SYSTEM.EXTEND.REMOVE command. A list of the keyword relationships currently in the control tables appears.
4. Enter **S** in the SEL field of the keyword relationship(s) you want to remove, then press enter. A message indicating the success of the remove appears.

   Note: Selecting a keyword from the Remove window will remove that keyword from the control tables without affecting the data in the PLATINUM Repository Extend model.
5. Use the END command (press PF3) to return to the KEYWORD Edit screen.

Extend Keyword Reference Tables

The following table contains keywords used for repository administration. All of these keywords are used exclusively for the Extend facility.

Keyword	Usage	Map	Description
ATTR	EXTEND	001	Attribute description
MAP ATTR	EXTEND	002	Map attribute relationship
ATTRCODE	EXTEND	003	Attribute code table association
COMMAND	EXTEND	004	Command description
CODE TBL	EXTEND	005	Code table definition
CODE VAR	EXTEND	006	Code table to variance association
DIALOG	EXTEND	007	Dialog definition
DLG VIEW	EXTEND	008	Dialog to map association
DLG PRIV	EXTEND	009	Dialog to privilege association.
META ENT	EXTEND	010	Entity type definition
MAP ENT	EXTEND	013	Map to entity association
PLCY SRC	EXTEND	014	Policy to entity source association

Keyword	Usage	Map	Description
PLCY TGT	EXTEND	015	Policy to entity target association
MESSAGE	EXTEND	016	Message definition
HLP ATTR	EXTEND	017	Attribute help
HLP CMD	EXTEND	018	Command help
HLP CODE	EXTEND	019	Code table help
HLP ENT	EXTEND	020	Entity help
HLP MSG	EXTEND	021	Message help
SCR LIT	EXTEND	023	Vertical screen literal relationship
MAP	EXTEND	024	Map definition
MAP LIKE	EXTEND	025	Clone map association
MIG PATH	EXTEND	026	Migration path relationship
PRIV	EXTEND	028	User privilege definition
PRIV CMD	EXTEND	029	Privilege to command association
PRIV ENT	EXTEND	030	Privilege to entity association
PRIV STS	EXTEND	031	Privilege to status association
USERPRIV	EXTEND	032	Privilege to user association
WORKPRIV	EXTEND	033	Privilege to workstation association
LITERAL	EXTEND	034	Vertical literal definition
CMD FOR	EXTEND	035	Command to entity association
ENT MIG	EXTEND	036	Entity to migration path association
SET OF	EXTEND	037	Set to entity association
SOURCE	EXTEND	038	Entity to source entity association
STATUS	EXTEND	039	Status definition
TARGET	EXTEND	040	Entity to target association
USER	EXTEND	041	User definition
KEYWORD	EXTEND	043	Keyword definition
VARIANCE	EXTEND	045	Code variance definition
POLICY	EXTEND	046	Entity policy definition
PLCY ENT	EXTEND	047	Policy to entity association

Chapter 12: Working with the Repository I/O Module

This chapter explains how to use the repository I/O module to manipulate data; it includes instructions about how to call the I/O modules or call the APIs directly.

This section contains the following topics:

The I/O Module (see page 149)
The I/O Module Calling Pattern (see page 150)
Special Considerations (see page 159)
Repository Control Tables (see page 165)
Sample Calls to the I/O Module (see page 166)
Stored Procedure Example (see page 170)

The I/O Module

The following sections briefly explain the I/O module and I/O maps.

What Is the Repository I/O Module?

The Repository I/O module:

- Performs the following actions for entities, relationships, associations, and associated extended text by calling the repository APIs:
  - INSERT
  - UPDATE
  - REPLACE
  - DELETE
  - SELECT

  Note: These functions work on one single instance at a time.
- Supports calls that include a MAP_ID used to format or retrieve information from the data block
- Has data passed to and from the I/O module in a 4 KB (4096 byte) block
- Supports up to fifteen simultaneously opened cursors
- Processes all the supplied values for the repository cross reference tables and the entity data tables
- Performs all repository security checking and validation

Why Use the Repository I/O Module?

Use the Repository I/O module to manipulate and retrieve repository data.

What Do You Use I/O Maps For?

Use Repository maps to pass data to and from the I/O module.

What Do I/O Maps Contain?

Each map contains generic information about how the I/O module processes data, including:

- Whether logging is used
- If an exit program is to be called
- The data attributes that the I/O module is to act on

I/O maps determine where data elements are stored in the block. A map that contains source and/or target entity information can be used to insert or update relationships and associations. For relationship data types, data can be selected from the source or target entity data tables, the cross reference table, as well as the relationship data table. All source and/or target entity attributes, with the exception of the Name, Status, and Version key are ignored for these actions. For associations, data can be selected from the source or target entity table or the cross reference table.

Only data for a single entity type is processed in each call (that is, data for the components of a relationship type cannot be processed as a group).

The I/O Module Calling Pattern

The repository I/O program is the interface to all data stored in the repository. All requests for data stored in the repository must be executed by calling the I/O module or by calling the APIs directly. Only the repository control tables can be accessed via direct SQL calls. The member COBOLIO of the SAMPPGM data set is a sample call to the I/O module from a COBOL program. Examples of API calls are shown later in this chapter along with the description of the various APIs.

The calling pattern for the I/O module is:

I/O Module	Calling Pattern
PL/I	DBXVLIO(IO_VARS, BLOCK, MAPAREA, SQLAREA, WHERE_DATA,ORDER_BY,

I/O Module	Calling Pattern
	RETURN_CODE,REASON,MSG_QUAL,MSG_ID);
COBOL	CALL 'DBXVLI2' USING IO-VARS, BLOCK-GROUP, MAP, SQL-AREA, WHERE-DATA, ORDER-BY, IO-RET-CODE, REASON, MSG-QUAL, MSG-ID.

Each of these variables and components are described in the sections below. The record layouts required by the I/O module are provided with each variable description in PL/I and COBOL.

The IO_VARS Variable

The IO_VARS variable structure contains values needed by the I/O module to control processing. Each element in the structure is explained in the Adding Values to the IO_VARS Variable Elements section.

I/O Module	Layout
PL/I	DCL 1 IO_VARS, 3 ACTION CHAR(1) INIT(' '), 3 CHECKPOINT_FLAG CHAR(1) INIT('Y'), 3 DOMAIN_SELECT CHAR(1) INIT('N'), 3 PROCESS_TEXT CHAR(1) INIT('N'), 3 BATCH_SW CHAR(1) INIT('N'), 3 LOGGING CHAR(1) INIT(' '), 3 TSO_LOGON CHAR(8) INIT(' '), 3 DIALOG CHAR(8) INIT(' '), 3 KEY CHAR(15) INIT(' '), 3 TEXT_FILE CHAR(46) INIT(' '), 3 MAP_ID FIXED BIN(31) INIT(0), 3 ENT_TYPE FIXED BIN(15) INIT(0), 3 CURSOR_NUM FIXED BIN(15) INIT(0), 3 RETURN_STMT_NO PIC'9999' INIT(0), 3 IO_DEBUG CHAR(1) INIT('N'), 3 VALIDATE CHAR(1) INIT('Y'), 3 SAVE_AUDIT CHAR(1) INIT('N'), 3 WORKSTATION_ID CHAR(15) INIT(' '), 3 RESERVED1 CHAR(1) INIT(' '), 3 RESERVED2 CHAR(1) INIT(' '), 3 RESERVED3 CHAR(1) INIT(' '), 3 RESERVED4 CHAR(50) INIT(' '), 3 RESERVED5 CHAR(50) INIT(' '), 3 RESERVED6 CHAR(50) INIT(' ');

I/O Module	Layout
COBOL	01 IO-VARS. 03 ACTION-IO PIC X(1) VALUE ' '. 03 CHECKPOINT-FLAG PIC X(1) VALUE 'Y'. 03 DOMAIN-SELECT PIC X(1) VALUE 'N'. 03 PROCESS-TEXT PIC X(1) VALUE 'N'. 03 BATCH-SW PIC X(1) VALUE 'N'. 03 LOGGING PIC X(1) VALUE ' '. 03 TSO-LOGON-IO PIC X(8) VALUE ' '. 03 DIALOG PIC X(8) VALUE ' '. 03 KEY-IO PIC X(15) VALUE ' '. 03 TEXT-FILE PIC X(46) VALUE ' '. 03 MAP-ID PIC S9(9) COMP VALUE 0. 03 ENT-TYPE-IO PIC S9(4) COMP VALUE 0. 03 CURSOR-NUM PIC S9(4) COMP VALUE 0. 03 RETURN-STMT-NO PIC 9(4) VALUE 0. 03 IO_DEBUG PIC X(1) VALUE 'N'. 03 VALIDATE PIC X(1) VALUE 'Y'. 03 SAVE-AUDIT PIC X(1) VALUE 'N'. 03 WORKSTATION-ID PIC X(1) VALUE ' '. 03 FILLER PIC X(1) VALUE ' '. 03 FILLER PIC X(1) VALUE ' '. 03 FILLER PIC X(1) VALUE ' '. 03 FILLER PIC X(50) VALUE ' '. 03 FILLER PIC X(50) VALUE ' '. 03 FILLER PIC X(50) VALUE ' '.

The IO_VARS Variable Elements

Enter values for the IO_VARS variable elements that will be used by the I/O module to control processing as follows:

I/O Vars	Enter
ACTION	C-To close a cursor. Cursors are automatically closed when no rows are left on a fetch. D-To delete an entity or text F-To fetch one entity from a currently open cursor I-To insert an entity or text L-To rollback M-To commit O-To open cursor in preparation of FETCHing many entities R-To replace an entity or text S-To select or retrieve a single entity extended text or

I/O Vars	Enter
	Domain values from the repository. Any values currently in the BLOCK variable are used to format the where clause of the Select statement. S-To select or retrieve a single entity extended text or Domain values from the repository. Any values currently in the BLOCK variable are used to format the where clause of the select statement. U-To update an entity or text
CHECKPOINT_ FLAG	Y-If the I/O module should perform commits or rollbacks. N-If the I/O module should not perform commits or rollbacks. **Note:** If set to Y, the I/O module automatically commits or rolls back each insert, update, delete, or replace. If set to N, the calling program is responsible for issuing commits or rollbacks, or to call the I/O module to do them.
DOMAIN_ SELECT	Y-To indicate that domain values will be selected. N-If you do not want a select request used to retrieve Domain values. **Note:** The specified map must contain a table name and prefix for domain values or an error will result.
PROCESS_ TEXT	Y-If you want the action performed on the Extended Text for the entity. N-If you do not want the action performed on the Extended Text for the entity. **Note:** The specified map must identify a table name and prefix for the text data or an error results. The respective text type is retrieved if the value is 1-5.
LOGGING	Y-To force logging on. N-To force logging off. Leave this field blank to use the logging specified on this map.
BATCH_SW	Y-To indicate that the call is for a batch program. N-To indicate that the call is an on-line call.
TSO_LOGON	The USERID used by the I/O module to determine if the repository security will allow the requested action.
DIALOG	The current dialog. **Note:** The default is DBEXCEL.
TEXT_FILE	The name of a file to use for text processing. **Note:** This file must have an LRECL of 80. If left blank, the I/O uses the file currently allocated to the DDNAME DBXTEXT as a default.

I/O Vars	Enter
MAP_ID	The ID of the map to use to process the request.
ENT_TYPE	The Internal identifier for the entity currently being processed.
CURSOR_NUM	The number of the cursor to use if the action is SELECT, OPEN, FETCH, or CLOSE. **Note:** Valid values are 1 - 15. If you enter a value other than 1 through 15, 1 is entered by default.
IO_DEBUG	Y-To have the I/O module print trace information to the SYSPRINT DD. N-If you do not want the I/O module to print trace information to the SYSPRINT DD.
VALIDATE	Y-To have validation performed. N-To bypass validation. **Note:** You must have access to the BLOAD.NOVALID command to bypass validation.
WORKSTATION_ID	The workstation to use for processing workstation groups.
SAVEAUDIT	Y-To have all audit attributes set to the values from the SQL statement or data file rather than from the current time, date, or user ID. N-If you do not want to have all audit attributes set to the values from the SQL statement or data file.
RESERVED 1-6	Reserved for future use.
RETURN_STMT_NO	A statement number for DB2 statements executed in the program. **Note:** If an SQL error occurs, this holds the statement number where the error occurred.

The BLOCK Variable

The BLOCK variable refers to the data area where the entity attribute is stored. This variable is declared as a contiguous block of 4096 bytes, with the following layout:

I/O Module	Layout
PL/I	DCL1 BLOCK UNALIGNED, 3 ENT_TYPE FIXED BIN(15) INIT(0), 3 ENT_ID FIXED BIN(31) INIT(0),

```
                3 SOURCE_ID   FIXED BIN(31)   INIT(0),
                3 TARGET_ID   FIXED BIN(31)   INIT(0),
                3 LOCK_ID     FIXED BIN(31)   INIT(0),
                3 TSO_LOGON   CHAR(8)         INIT(' '),
                3 TIMESTAMP   CHAR(26)        INIT(' '),
                3 DATA        CHAR(4044)      INIT(' ');
```

```
COBOL       01 BLOCK.
              05 X-REF-GROUP.
                10 ENT-TYPE    PIC S9(4) COMP VALUE 0.
                10 ENT-ID      PIC S9(9) COMP VALUE 0.
                10 SOURCE-ID   PIC S9(9) COMP VALUE 0.
                10 TARGET-ID   PIC S9(9) COMP VALUE 0.
                10 LOCK-ID     PIC S9(9) COMP VALUE 0.
                10 TSO-LOGON   PIC X(0008) VALUE ' '.
                10 TIMESTAMP   PIC X(26)  VALUE 0.
              05 DATA       PIC X(4044) VALUE ' '.
```

Note: Some values are predefined at the start of the block. These fields are the columns on the DBX_XREF table and are used for every action, except for the ENT_TYPE column. ENT_TYPE is retrieved for SELECTs and FETCHes. The ENT_TYPE from the IO_VARS structure is used by the I/O module in all DB2 calls.

All data requested from the entity is returned in the variable. The position (offset) in the data variable is determined by the offset specified in the map definition. Any data in the block when a SELECT or OPEN is requested is used to format the where clause of the select statement. Only rows that have values equal to the values in the block are selected.

The data is returned in the same format as it is stored in DB2. For example, an INTEGER column is retrieved into 4 bytes in its binary format. Although any column type can be retrieved, the calling program is responsible for performing the conversions. The exception to this rule is decimal columns. Decimal values are always selected into a DECIMAL(15,4) value that takes up 8 bytes within data area. The calling program can then move the value to another decimal field if necessary.

Note: Make sure you do not modify the TIMESTAMP values after a SELECT, since it is used to control concurrence issues during UPDATEs.

The WHERE_DATA Variable

The WHERE_DATA variable refers to additional where criteria that are used during SELECT or OPEN actions. Any specified criteria is added to the where data formatted for values in the BLOCK.

Note: Do **not** include the keywords WHERE or AND as the first values in the data since they are added automatically.

Prefix all columns for the entity storage table with **E**. Prefix all columns of the source and target entities of a relationship with **S** (for Source) and **T** (for Target).

I/O Module	Layout
PL/I	`DCL 1 WHERE_DATA  CHAR(4000) INT(' ');`
COBOL	`WHERE-DATA  PIC X(4000) VALUE ' '.`

The ORDER_BY Variable

The ORDER_BY variable refers to a list of columns used in the order by clause for a select or open. This overrides any order by that is already defined as part of the map.

Note: Do **not** include the keywords ORDER BY since they are added automatically.

Prefix all columns as follows:

- For the entity storage table, use the prefix E
- The source and target entities of a relationship should be prefixed with an S (for Source) and T (for Target)

I/O Module	Layout
PL/I	`DCL 1 ORDER_BY  CHAR (250) INIT(' ')`
COBOL	`01 ORDER-BY  PIC X(0250)VALUE ' '.`

The MAPAREA Variable

The MAPAREA variable refers to the area to use to store the repository map. The size required is (n * 44) + 1143 where n is the number of columns to retrieve. The maximum size is 14343 to select 300 columns.

Note: Never modify the value by the calling program or the results will be unpredictable (and possibly damaging).

Use the map area to select the map definition for the control tables. When processing multiple entities with the same map, the overhead of rebuilding the map definition is saved, this substantially improves performance.

I/O Module	Layout
PL/I	`DCL 1 MAPAREA CHAR(14143) INIT (' ');`
COBOL	`01 MAPAREA PIC X(14143) VALUE ' '.`

The SQLAREA Variable

The SQLAREA variable refers to the area used by the I/O module to communicate with DB2. The size required is (*n* * 44) + 16 where *n* is the number of columns to retrieve. The maximum size needed for 300 columns is 13216 bytes.

I/O Module	Layout
PL/I	`DCL 1 SQLAREA CHAR(13216) INIT(' '):`
COBOL	`01 SQLAREA PIC X(13216) VALUE ' '.`

The REASON Variable

The REASON variable is a list of variables that contains any tokens relating to an error.

I/O Module	Layout
PL/I	`DCL 1 REASON CHAR(80) INIT(' ');`
COBOL	`01 REASON PIC X(80) VALUE ' '.`

The MSG_ID Variable

The MSG_ID variable refers to the message ID for any error that occurred. For DB2 errors, the ID is the absolute value of the SQLCODE. The I/O module never calls the message program to display any errors. The display of messages must be ensured by the calling program.

I/O Module	Layout
PL/I	`DCL 1 MSG_ID  FIXED BIN(31) INIT(0);`
COBOL	`01 MSG-ID  PIC S9(9) COMP VALUE 0.`

The RETURN_CODE Variable

The RETURN_CODE variable is the standard repository return code.

I/O Module	Layout
PL/I	`DCL 1 RETURN_CODE  IXED BIN(31) INIT(0);`
COBOL	`01 RETURN-CODE  PIC S9(9) COMP VALUE 0.`

Possible values include:

Return Code	Indicates
0	A successful execution.
4	Not found for FETCH or SELECT. On FETCH, the respective cursor is closed automatically.
8	An SQL error has occurred. For more information, the calling program should view the MSG_ID, MSG_QUAL, and REASON.
12	An error occurred which prevents the requested action from completing. For more information, the calling program should view the MSG_ID, MSG_QUAL, and REASON.
16	A severe error occurred or the passed variables cannot be used. For more information, the calling program should view the MSG_ID, MSG_QUAL, and REASON.

The MSG_QUAL Variable

The MSG_QUAL variable refers to the qualifier of the message. The I/O module only returns XIO, SQE, or SQW messages. Other message qualifiers may be passed back from the I/O module if they came from a subroutine called by the I/O module (the exit program, cockeyed validation program or code table validation program).

I/O Module	Layout
PL/I	`DCL 1 MSG_QUAL  CHAR(03) INIT(' ');`
COBOL	`01 MSG-QUAL  PIC X(03) VALUE ' '.`

Special Considerations

The following sections explain special considerations when using the I/O module.

I/O Module Maps

I/O module maps determine where attribute values are located in the data block.

Because the I/O module checks security for each action:

- All entity maps and relationship maps must contain the STATUS for the entity type itself
- Association maps or relationship maps must contain the STATUS attribute for the source entity type

  Note: If the maps do not contain this information, the security privilege checks may allow an action even if the actual security does not exist.
- Maps that define entities or relationships must have an entity table name defined
- Maps that define associations or relationships must have source and target table names defined
- Map used for domain processing must contain a domain table name

Open-Close-Fetch Actions

When selecting or using the open-fetch-close actions, the I/O module can support up to fifteen open cursors at the same time. Since the select action also uses a cursor, a select is equivalent to open-fetch-close with only one fetch.

Note: Make sure to set the CURSOR_NUM variable correctly before each select, open, fetch, or close to avoid any DB2 errors. Even though the select action uses a cursor, it returns an SQL error of -811 (MSG_QUAL = SQE and MSG_ID = 811) if multiple rows are available to satisfy the request. In this case, the I/O module returns only the first row that met the criteria.

The repository security is called during select and fetch operations. Only rows to which the user has access are returned. If any rows are not returned due to security, the MSG_ID and MSG_QUAL are set, but the return code will be 0 if any rows are returned, or 4 if none are returned. The calling program must decide if it will display the message.

Message Processing

The I/O module will never call the message program to display any errors that occur. The calling program determines if or how to call the message program based upon the values set by the I/O module. The I/O module formats the message variables based on the information that it has available.

Map Selection and Performance Impact

The I/O module selects the map layout from the control tables whenever the requested map is not already in the MAPAREA variable. To avoid this overhead, calls that use the same map should be grouped together or multiple map areas be maintained by the calling program.

Since the map area can be much smaller for maps with less than 300 columns, it is much more efficient to use multiple small maps than to accept the overhead of the select of the map for each call. The minimum required bytes for a map is (*n* * 44) + 1143 where *n* is the number of columns. Only PL/I programs can use smaller MAPAREAs, as COBOL does not support dynamically sized variables.

SQLAREA Size

The SQLAREA can be used by all of the active cursors open at the same time. There is no need to ever have multiple SQLAREAS defined. The minimum required bytes for the SQLAREA is (*n* * 44) + 16 where *n* is the number of columns. COBOL programs must declare the size to be 13216, since COBOL does not support dynamically sized variables.

Decimal Fields

Decimal fields are handled in a specific manner for communication with the I/O module. All decimal fields are passed to and from the I/O module in the BLOCK as 8 bytes defined as FIXED DEC(15,4) [or COBOL 9(11)V9(4) COMP-3]. The actual DB2 column can be defined as anything with 11 or fewer digits to the left of the decimal and 4 or less to the right of the decimal. The calling program must then determine if it is necessary to move the value from the BLOCK for calculation purposes.

The Where Clause on SELECT or OPEN

When performing a SELECT or OPEN, the where clause of the Select statement is formatted by checking all values in the BLOCK variable to determine if their corresponding bytes in the BLOCK are blank. If they are not, the value is included as part of the where clause. Cross reference fields that are included at the start of the BLOCK are checked to see if they are zero for integer, or blank (for character and timestamp) before including them in the where clause (except for ENT_TYPE as noted above).

The values in the BLOCK.DATA are checked to see if they do not equal spaces to determine if they will be used (setting BLOCK.DATA = ' ' causes no WHERE clause to be added for the variables in the BLOCK.DATA). The WHERE_DATA supplied is used in addition to anywhere criteria formatted in this manner (care must taken to *blank out* all fields in the block, especially between multiple calls to the I/O).

When processing extended text, only the ENT_ID is used to select the text. All other data in the BLOCK is ignored. When performing a domain select, only the columns identified as domain key columns in the map (domain use = "K") are used to format the select from the domain table. If their corresponding values in the BLOCK are blank, they are not used to format the where clause.

Special "X" (Excluded) Attributes

The map definition can include attributes that do not represent actual columns on the entity data table. These attributes are defined by an "X" for the ATTR_CODE in the map. These attributes are then defined to have the same offset as a *real* attribute, which enables the user to view a very long column as many shorter pieces, or allow separate validation on subsets of a field (the first 4 bytes validated against one code table and the second three bytes against another code table). Any fields used for this purpose must be character fields.

UPDATE Statements

When calling for an UPDATE action, only the ENT_ID, the TIMESTAMP from the BLOCK, and the ENT_TYPE from the IO_VARS are used in the Where clause. The TSO_LOGON and TIMESTAMP are the only columns updated on the cross reference table. TIMESTAMP is set to CURRENT TIMESTAMP and TSO_LOGON to the TSO_LOGON in the IO_VARS.

TIMESTAMP should never be changed, or the row will not be found for the UPDATE (an SQL error could also result if the value is not a valid timestamp). The TIMESTAMP column and TSO_LOGON are used for concurrent processing. If either is changed after a SELECT, the UPDATE fails. In addition, entities that are locked cannot be updated.

Columns from the data table described in the map are updated if they have a format of CHAR, VARCHAR, DATE, TIME, TIMESTAMP, DECIMAL, INTEGER, or SMALLINT. Character fields can be set to the TSO_LOGON from the IO_VARS if the ATTR_CODE in the map is set to *C*. Likewise, attributes with a Date, Time, or Timestamp data type can be set to the current value special registers through the DB2 built-in functions by setting the ATTR_CODE to *C.*

DELETE Statements

When calling for delete, only the ENT_ID is used to identify the entity to delete. All relationships the entity participates in are deleted and all *chained* relationships for an infinite number of levels are also deleted (limited only by available space). In order to delete all of the chained relationships, the I/O module must be able to find a table name and prefix for each entity type it encounters in the chain. This is only possible if the current dialog has a map corresponding to each entity type encountered in the chain.

If a map cannot be found for delete, an error results and the delete is not performed. An entity locked to a deleted entity (including those that are part of the chain) are unlocked (LOCK_ID set to 0). A locked entity cannot be deleted until the lock is removed.

INSERT Statements

When performing an INSERT the entity ID is set to 1 + the maximum entity ID. In addition, all values from the block are inserted into the tables. The exceptions to these generalizations are ENT_TYPE (which comes from the IO_VARS), TIMESTAMP (which is set to CURRENT TIMESTAMP), LOCK_ID (which is set to 0), and TSO_LOGON (which is set to the value in the IO_VARS).

Only columns with a format of CHAR, VARCHAR, DATE, TIME, TIMESTAMP, DECIMAL, INTEGER, or SMALLINT are inserted. The system also checks for duplicate entities, violation of maximum restrictions, policies, and dependencies before the INSERT action is processed.

REPLACE Statements

The REPLACE action allows one entity to replace another. It works as follows:

- All relationships and associations that use the entity to be replaced as their source are deleted
- All relationships and associations that use it as their target are updated to point to the replacement entity
- The ENT_ID of the BLOCK is the entity to be replaced
- The LOCK_ID is the entity that is replacing it
- The entity to be replaced must not be locked (unless it is locked to the replacement entity) or the REPLACE fails

FETCH

Use FETCH to select the next row in a set of rows meeting the search criteria specified in an OPEN. On a FETCH action, the return code is set to 0 if the action was successful or to 4 if no new data is returned. When the return code is set to 4, the cursor is automatically closed, so there is no need to call the I/O module to close it. The FETCH used by the I/O module is similar to the DB2 FETCH.

COMMIT and ROLLBACK

The I/O module does not perform any COMMITs or ROLLBACKs when doing open, fetch, close, or select. The I/O module will ROLLBACK or COMMIT after all insert, update, delete, or replace calls if the checkpoint flag is set to **Y**. The calling program must take care to manage its committing and rolling back to ensure that the I/O module does not inadvertently commit or rollback any changes made by the calling program.

Any commits or rollbacks made by the calling programs close any currently open cursors. Using the I/O module to perform any commits or rollbacksl ensures that the PLATINUM repository log file is correct.

LOGGING

The I/O module can log all statements that it performs. Only INSERT, UPDATE, DELETE, COMMIT and ROLLBACK are logged. The format of the log file is a variably blocked file with an LRECL of 4092 and a blocksize of 4096. The I/O module writes out all statements it performs to the file, including commits and rollbacks. Each user must have their own log file allocated. The DDNAME for the file is DBXLOG. If no file is allocated to DBXLOG, no logging is performed.

EXIT Routine

The I/O module can call a compiled exit subroutine before any action is taken by the I/O module. The program, named DBXEXIT, is called with the parameters (IO_VARS, BLOCK, RETURN_CODE, REASON, MSG_QUAL, and MSG_ID) that correspond to the parameters passed to the I/O module. The exit routine can then perform any action based on the IO_VARS. If the RETURN_CODE is greater than 0, the I/O module will not perform any action, preserving the return code and message information from the exit. A sample program (SMPEXIT) is provided.

Whether the exit routine is called or not depends on a parameter stored with each map. The exit routine is called before any validation is performed and before any I/O processing is performed (that is, before any values are retrieved on a select). The exit routine is called for the actions Open, Select, Fetch, Update, Insert, Delete, and Replace when not processing domains. In order to use the new exit, you must relink the CA Repository program with any new exit that is written.

Entity Sets

Entity sets are not actual entity types, but a logical grouping of other entity types. They are defined to the CA Repository, but no entities actually exist for a set type. All entities in a set type are stored on the same data table, and the name, status, and version will be unique across all entity types in the set.

Whenever the I/O module is called for a select operation (S or O) and the entity type is a set, the rows retrieved are from any entity type in the set. The I/O module should never be called with a set entity type for insert, update, delete, or replace.

Note: Due to size and performance considerations, there is a maximum of ten entity types per set.

Extended Text

All extended text processing performed through the I/O is done using a file that contains the text. When inserting text, the entity must not have any text or an error results. UPDATE appends any new text to the end of any existing text. REPLACE replaces existing text with the new text. The tables used to store the extended text must have the following three columns: ENT_ID (integer); SEQ_NUM (Smallint); and TEXT (Character 80) or errors will result.

Policies

Whenever an entity is inserted or updated, it can be validated against a *policy* to allow more flexibility in preventing undesirable definitions from being entered into the repository. Each entity type can have three policies:

- An entity policy to validate the entity
- A source policy to validate the source entity of relationships and associations
- A target policy to validate the target of a relationship or association

The source and target policies are used when inserting a relationship or association to force the source or target entity to meet some predefined criteria. The policy is not checked when either the source or target entity is updated outside of the relationship or association in which the policy is defined.

Repository Control Tables

The I/O module uses the repository map control tables listed below to perform all processing.

Control Table	Description
DBX_IO_MAP_DESC	A table that contains the information about where the entity data is stored and where the source and target entity data are stored for relationships and associations.
DBX_IO_MAP_ATTR	A table that stores all of the attributes of the entity type that are contained in the map and information on where they are placed in the data block.
DBX_ENT_TYPE_DESC	An entity type description table that contains all information about an entity type, including special validations to be performed for the entity type.
DBX_DIALOG	A dialog table used to determine which maps are used for a given entity type.

Control Table	Description
DBX_POLICY	A table that contains the policies defined for an entity type.
DBX_SET_TYPE	A table that contains the entity types that make up a set entity. A maximum of ten entity types per set has been imposed due to space and performance considerations.

Sample Calls to the I/O Module

The following pages contain examples of calls to the repository I/O module.

Sample Map Definition

The following is an example of a map definition.

```
 ---------------- SAMPLE DECLARTION FOR PRE-DEFINED MAP ----------
 -- THIS EXAMPLE SHOWS HOW A MAP WOULD BE DEFINED FOR A KNOWN   --
 -- MAP LAYOUT. THIS STRUCTURE REDEFINES THE BLOCK, SO IT       --
 -- OCCUPIES THE SAME PHYSICAL STORAGE SPACE. IN SOME           --
 -- PROGRAMS MANY RECORD LAYOUTS LIKE THIS WILL REDEFINE THE    --
 -- BLOCK AREA, ONE FOR EACH MAP USED BY THE PROGRAM.           --
 -- NOTE: CARE MUST BE TAKEN WHEN DEFINING THE MAP OFFSETS      --
 -- BECAUSE THE "ALIGNED/UNALIGNED" ATTRIBUTE OF PL/I AND       --
 -- THE "SYNCHRONIZED" ATTRIBUTE FOR COBOL WILL CHANGE          --
 -- THE PHYSICAL LAYOUT OF THE DATA STRUCTURE                   --
DCL 1 ENTITY_BLOCK UNALIGNED DEFINES (BLOCK),
      3 XREF_FILLER                    CHAR(52),
      3 NAME                           CHAR(50),
      3 STATUS                         CHAR(08),
      3 VERSION                        CHAR(06),
      3 DESCRIPTION                    CHAR(72),
      3 VALUE1                  FIXED BIN(15),
      3 VALUE2                         CHAR(05),
      3 VALUE3                         CHAR(22),
      3 END_FILLER                     CHAR(3879);

-------------  COBOL RECORD LAYOUT --------------------
      01  ENTITY-BLOCK REDEFINES BLOCK.
          05  XREF-GROUP                PIC X(52).
          05  NAME                      PIC X(50).
          05  STATUS                    PIC X(08).
          05  VERSION                   PIC X(06).
          05  DESCRIPTION               PIC X(72).
          05  CURSOR-NUM                PIC S9(4) COMP.
          05  VALUE2                    PIC X(05).
          05  VALUE3                    PIC X(22).
          05  FILLER                    PIC X(3879).
```

Sample Select for a Known Entity ID

The following is an example of a Select for a known Entity ID.

```
 -----------  SAMPLE SELECT FOR KNOWN ENT_ID -----------
 -- INITIALIZE THE BLOCK VALUES. BLANK OUT ALL VALUES EXCEPT
 -- TYPE AND ID SO THAT THEY ARE THE ONLY VALUES USED TO
 -- PERFORM THE SELECT
   BLOCK.ENT_ID    = HOLD_ENT_ID;
   BLOCK.ENT_TYPE  = HOLD_ENT_TYPE;
   BLOCK.DATA      = ' ';
   BLOCK.SOURCE_ID = 0;
   BLOCK.TARGET_ID = 0;
   BLOCK.LOCK_ID   = 0;
   BLOCK.TSO_LOGON = ' ';
   BLOCK.TIMESTAMP = ' ';
 -- INITIALIZE THE IO_VARS VALUES
   IO_VARS.MAP_ID          = HOLD_MAP_ID;
   IO_VARS.ACTION          = 'S';            -- SELECT AN ENTITY
   IO_VARS.PROCESS_TEXT    = 'N';
   IO_VARS.DOMAIN_SELECT   = 'N';
   IO_VARS.CHECKPOINT_FLAG = 'Y';
   IO_VARS.TEXT_FILE       = ' ';
   IO_VARS.TSO_LOGON       = HOLD_TSO_LOGON;
   IO_VARS.ENT_TYPE        = HOLD_ENT_TYPE;
   IO_VARS.DIALOG          = HOLD_DIALOG;
-- DO NOT SPECIFY ANY ADDITIONAL WHERE CRITERIA OR ORDER BY
   WHERE_DATA              = ' ';
   ORDER_BY                = ' ';
   RETURN_CODE = 0;
   CALL DBXVLIO(IO_VARS,BLOCK,MAP,SQLAREA,WHERE_DATA,ORDER_BY,
                RETURN_CODE,REASON,MSG_QUAL,MSG_ID);
-- IF THERE IS AN ERROR, DISPLAY THE MESSAGE
   IF RETURN_CODE ^= 0 THEN
     DO;
     HOLD_STMT_NO = RETURN_STMT_NO;
     HOLD_MODULE  = 'DBXVLIO';
     CALL DISPLAY_MESSAGE;
     END;
```

Sample Select for a Known Map (1 of 2)

The following is an example of part one of two of a Select for a known map.

```
 -----------  SAMPLE SELECT USING KNOWN MAP ------------------
 -- THIS WILL SELECT ALL ENTITIES WITH A NAME OF "TEST" AND --
 -- VALUE1 = 90                                             --
@JCL =
 -- BLANK OUT ALL OF THE BLOCK VALUES BEFORE FORMATTING WITH
 -- THE DESRIED VALUES.
   BLOCK.ENT_ID    = 0;
   BLOCK.ENT_TYPE  = HOLD_ENT_TYPE;
   BLOCK.DATA      = ' ';
   BLOCK.SOURCE_ID = 0;
   BLOCK.TARGET_ID = 0;
   BLOCK.LOCK_ID   = 0;
   BLOCK.TSO_LOGON = ' ';
   BLOCK.TIMESTAMP = ' ';
 -- USE THE PREDEFINED RECORD LAYOUT TO PLACE VALUES INTO THE
 -- BLOCK VARIABLE. THESE VALUES WILL BE USED BY THE I/O
 -- FORMAT THE SELECT STATEMENT
   ENTITY_BLOCK.NAME = 'TEST';
   ENTITY_BLOCK.VALUE1 = 90;
 --
   IO_VARS.MAP_ID          = HOLD_MAP_ID;
   IO_VARS.ACTION          = 'O';          -- OPEN A CURSOR
   IO_VARS.PROCESS_TEXT    = 'N';
   IO_VARS.DOMAIN_SELECT   = 'N';
   IO_VARS.CHECKPOINT_FLAG = 'Y';
   IO_VARS.TEXT_FILE       = ' ';
   IO_VARS.TSO_LOGON       = HOLD_TSO_LOGON;
   IO_VARS.ENT_TYPE        = HOLD_ENT_TYPE;
   IO_VARS.DIALOG          = HOLD_DIALOG;
   WHERE_DATA              = ' ';
   ORDER_BY                = ' ';
   RETURN_CODE = 0;
   CALL DBXVLIO(IO_VARS,BLOCK,MAP,SQLAREA,WHERE_DATA,ORDER_BY,
                RETURN_CODE,REASON,MSG_QUAL,MSG_ID);
```

Sample SELECT Using a Known Map (2 of 2)

The following is an example of part two of two of a Select for a known map.

```
IF RETURN_CODE ^= 0 THEN
    DO;
    HOLD_STMT_NO = RETURN_STMT_NO;
    HOLD_MODULE  = 'DBXVLIO';
    CALL DISPLAY_MESSAGE;
    END;
-- CONTINUE FETCHING WHILE RETURN_CODE = 0. RETURN CODE WILL
-- BE SET TO 4 IF NO MORE ROWS EXIST
  DO WHILE(RETURN_CODE = 0);
    IO_VARS.ACTION          = 'F';      -- FETCH THE NEXT ENTITY
    BLOCK.DATA = ' ';
    CALL DBXVLIO(IO_VARS,BLOCK,MAP,SQLAREA,WHERE_DATA,ORDER_BY,
                 RETURN_CODE,REASON,MSG_QUAL,MSG_ID);
-- RETURN CODES OF 0 AND 4 ARE EXPECTED, ANYTHING ELSE IS AN ERROR
    IF RETURN_CODE ^= 0 & RETURN_CODE ^= 4 THEN
      DO;
      HOLD_STMT_NO = RETURN_STMT_NO;
      HOLD_MODULE  = 'DBXVLIO';
      CALL DISPLAY_MESSAGE;
      END;
    ELSE
-- ONLY PROCESS AFTER A FETCH IF THE RETURN CODE IS 0. A 4 MEANS
-- THAT NO MORE ENTITIES WERE FOUND
      IF RETURN_CODE = 0 THEN
        CALL PROCESS_THIS_ENTITY;
  END; /* DO WHILE */
```

Stored Procedure Example

The following example shows how to call a stored procedure in COBOL and process a result set. The stored procedure in this example is fictitious and is only used for explanation purposes.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SPTEST.
ENVIRONMENT DIVISION.
CONFIGUATION SECTION.
INPUT-OUTPUT SECTION.
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
   01  BADCODE                 PIC +9(10) USAGE DISPLAY.
   01  ENT-ID                  PIC S9(9) USAGE COMP.
   01  ENT-TYPE                PIC S9(4) USAGE COMP.
   01  MAP-ID                  PIC S9(9) USAGE COMP.
   01  DIALOG                  PIC X(8).
   01  CASE-SEN                PIC X.
   01  NAME-WILDCARD           PIC X.
ENAME.
49 ENAMELEN             PIC S9(4) USAGE COMP.
49 ENAMETXT             PIC X(254).
ESTATUS.
49 ESTATUSLEN           PIC S9(4) USAGE COMP.
49 ESTATUSTXT           PIC X(8).
   01  EVERSION                PIC X(2).
   01  MSG-QUAL                PIC X(3).
   01  MSG-ID                  PIC S9(4) USAGE COMP.
   01  RTCODE                  PIC S9(9) USAGE COMP.
REASON.
49 REASONLEN            PIC S9(4) USAGE COMP.
49 REASONTXT            PIC X(80).
   01  OUT-NAME                PIC X(254).
   01  OUT-STATUS              PIC X(8).
   01  OUT-VERSION             PIC X(2).
```

```
*INCLUDE INDICATOR VARIABLES FOR EVERY FIELD
* PASSED TO AND FROM THE STORED PROCEDURE.

   77  ENTTYPEIND              PIC S9(4) COMP.
   77  ENTIDIND                PIC S9(4) COMP.
   77  MAPIDIND                PIC S9(4) COMP.
   77  DIALOGIND               PIC S9(4) COMP.
   77  CASESENIND              PIC S9(4) COMP.
   77  NAMEWILDCARDIND         PIC S9(4) COMP.
   77  ENAMEIND                PIC S9(4) COMP.
   77  ESTATUSIND              PIC S9(4) COMP.
   77  EVERSIONIND             PIC S9(4) COMP.
   77  RTCODEIND               PIC S9(4) COMP.
   77  MSGQUALIND              PIC S9(4) COMP.
   77  MSGIDIND                PIC S9(4) COMP.
   77  REASONIND               PIC S9(4) COMP.

  * INCLUDE A RESULT SET LOCATOR FOR THE RESULT SET
  * THAT IS RETURNED.
   01  LOC                  USAGE SQL TYPE IS
                             RESULT-SET-LOCATOR VARYING.

  * INCLUDE SQLCA
   EXEC SQL INCLUDE SQLCA END-EXEC.

   PROCEDURE DIVISION.
       MOVE ZERO TO RET-CODE.
       MOVE 12598 TO ENT-ID.
       MOVE 126 TO ENT-TYPE.
       MOVE 126 TO MAP-ID;
       MOVE ZERO TO ENTIDIND, ENTTYPEIND, MAPIDIND.

  *  SET NULL INDICATORS FOR UNUSED INPUT FIELDS AND ALL
  *    OUTPUT FIELDS
       MOVE -1 TO DIALOGIND, CASESENIND, NAMEWILDCARDIND,
              ENAMEIND, ESTATUSIND, EVERSIONIND,
              RTCODEIND, MSGQUALIND, MSGIDIND,
              REASONIND.

  *   CALL THE STORED PROCEDURE
       EXEC SQL
         CALL CRGETNSV(:ENT_TYPE INDICATOR :ENTTYPEIND,
                   :ENT-ID INDICATOR :ENTIDIND,
                   :MAP-ID INDICATOR :MAPIDIND,
                   :DIALOG INDICATOR :DIALOGIND,
                   :CASE-SEN INDICATOR :CASESENIND,
                   :NAME-WILDCARD INDICATOR :NAMEWILDCARDIND,
```

```
                    :ENAME INDICATOR :ENAMEIND,
                    :ESTATUS INDICATOR :ESTATUSIND,
                    :EVERSION INDICATOR :EVERSIONIND,
                    :RET-CODE INDICATOR :RTCODEIND,
                    :MSG-QUAL INDICATOR :MSGQUALIND,
                    :MSG-ID INDICATOR :MSGIDIND,
                    :REASON INDICATOR :REASONIND)
     END-EXEC.
    IF SQLCODE NOT EQUAL +466
*   BAD SQLCODE
          DISPLAY 'ERROR CALLING STORED PROCEDURE'
          MOVE SQLCODE TO BADCODE
          DISPLAY 'SQLCODE = ' BADCODE
          DISPLAY 'SQLERRMC = ' SQLERRMC
    ELSE
       IF RET-CODE NOT EQUAL 0
          DISPLAY 'ERROR CALLING STORE PROCEDURE'
          MOVE RET-CODE TO BADCODE
          DISPLAY 'RETURN CODE = ' BADCODE
          DISPLAY 'MSG-QUAL = ' MSG-QUAL
          MOVE MSG-ID TO BADCODE
          DISPLAY 'MSG-ID = ' BADCODE
          DISPLAY 'REASON = ' REASON
       ELSE
         PERFORM GET-RESULT-SET.
     GOBACK.

 GET-RESULT-SET.
     EXEC SQL
         ASSOCIATE LOCATORS (:LOC) WITH
              PROCEDURE CRGETNSV
     END-EXEC.
     EXEC SQL
         ALLOCATE NSVCUR CURSOR FOR RESULT SET :LOC
     END-EXEC.
     PERFORM GET-ROWS UNTIL SQLCODE = +100.
     EXEC SQL
         CLOSE NSVCUR
     END-EXEC.
 GET-ROWS.
     EXEC SQL
          FETCH NSVCUR INTO
               :OUT-NAME, :OUT-STATUS, :OUT-VERSION
     END-EXEC
     IF SQLCODE = 0
        DISPLAY 'NAME = ' OUT-NAME
        DISPLAY 'STATUS = ' OUT-STATUS
    DISPLAY  ' VERSION = ' OUT-VERSION.
```

Chapter 13: CA Repository z/OS or APIs

This section contains the CA Repository APIs and provides you with a description, input, output, explanations, and examples. These APIs consists of DB2 stored procedures and DB2 functions most of which can be called from any platform or language that supports ODBC or JDBC calls to DB2 stored procedures. The following repository APIs are provided:

API Name	Description
CGETVER	Get available version number
CRTXT80	Text processing
DBXIOD	Delete object
DBXIOR	Replace object
DBXIOM	Merge Object
DBXIOU	Update Object
DBXIOI	Insert Object
DBXIOS	Select Object
DBXSEC	Internal Security
DBXNSVI	Get Name, Status and Version
Get_streamtext_32K	Get text as one column
DBXIA	Run Impact Analysis
CMSGS	Build message as msg qual, msg id, reason

This section contains the following topics:

CGETVER (see page 174)
CMSGS (see page 177)
CRTXT80 (see page 178)
DBXENTID (see page 184)
DBXIA (see page 186)
DBXIOD (see page 189)
DBXIOI (see page 190)
DBXIOM (see page 195)
DBXIOR (see page 198)
DBXIOS (see page 200)
DBXIOU (see page 211)
DBXNSVI (see page 217)
DBXSEC (see page 219)
Get_streamtext_32K (see page 222)

CGETVER

CGETVER is a stored procedure that returns an available version number for a given object.

Input

The following table describes the input parameters for this procedure.

Field	Size	Description	Parameter Number
Entity type	Half word	Internal identifier for each entity definition This field is required.	1
Map ID	Full word	The map used to obtain the name, status and version attributes This field is required.	2
Action	1 char	L - Get the lowest version number available H - Get the highest version number available N - Get the next version number available This field is required.	3
Name	245 chars varying	The content of name attribute that will be used to search for an available version number	4
Status	8 chars varying	The content of the status attribute that will be used to search for an available version number	5

Output

The following table describes the fields processed by the CGETVER stored procedure:

Field	Size	Description	Parameter Number
Version	Half word	The available version number.	6
Return code	Full word	A return code other than zero indicates an error	7
Message qualifier	3 chars	The error message prefix	8
Message id	Full word	The message ID	9
Reason	80 chars varying	The short message text	10

Notes:

- The stored procedure is defined as General with NULLS
- An indicator variable must be set for all input and output parameters
- Set all output variable indicators to -1
- Set any unused input variable indicators to -1

  Setting output variable indicators and unused input variable indicators to -1 improves performance, as these fields are not transmitted to the stored procedure
- Set all used input variable indicators to 0

How the Program Determines the Available Version Number

Assume that the ELEMENTS table contains version 0, 1,10 and 50 for an element with the name of customer_name and a status of REP40.

Results are as follows:

- If the action is L, the lowest available version returned is 2
- If the action is N, the next available version returned is 5.
- If the action is H, the highest available version returned is 32767
- If the version returned is negative or greater that 32767, an error message with return code of 4 is returned

CGETVER Example

```
Get the next available version for Element Customer_name with a Status of REP40.
   eversionind = -1;
   rtcodeind = -1;
   msgqualind = -1;
   msgidind = -1;
   reasonind = -1;
   enttypeind = 0;
   mapidind  = 0;
   actionind = 0;
   enameind = 0;
   estatusind = 0;
   ename =  'Customer_name';
   estatus = 'REP40';
   action = 'N';
   ent_type = 204;
   map_id  = 204;
   exec sql
   call cgetver(
     :ent_type  INDICATOR    :ENTTYPEIND,
     :MAP_ID INDICATOR      :MAPIDIND,
     :action INDICATOR      :actionIND,
     :eNAME INDICATOR      :ENAMEIND,
     :eSTATUS INDICATOR    :ESTATUSIND,
     :VERSION INDICATOR   :EVERSIONIND,
     :ret_CODE INDICATOR    :RTCODEIND,
     :MSG_QUAL INDICATOR    :MSGQUALIND,
     :MSG_ID INDICATOR      :MSGIDIND,
     :REASON INDICATOR      :REASONIND) ;
  if sqlcode ^= 0 then
    do;
      return_code = sqlcode;
    end;
  else
    do;
      if ret_code ^= 0 then
         return_code = ret_code;
      else
         return_code = 0;
      end;
```

CMSGS

CMSGS stored procedure concatenates all MSG_TEXT from DBX_MSG_TEXT a given MSG_QUAL and MSG_ID and the following operations:

- Deletes all // occurances from beginning and end of the concatenated string
- Replaces all other // occurances(not at being or end) with \n character
- Deletes the following chars from the concatenated string =HI, =LO, =CB, =CE, =RB, =RE, =LB, =LE
- Replaces the variables from V1...V10 with the values provided in input parameter preason. Values in this parameter need to be divided based on delimeter provided in another input parameter pdellim.

Input

The following table describes the input fields processed by the CMSGS stored procedure:

Field	Size	Description	Parameter Number
Delimeter	Varchar(245)	It is used to divide the reason with the character provided here as the delimeter.	1

InOut

The following table describes the InOut fields produced by the CMSGS stored procedure:

Field	Size	Description
MSG QUAL	Char(3)	The message prefix.
MSG ID	Integer	The message ID.
REASON	Varchar(80)	The short message text.

Output

The following table describes the Output fields produced by the CMSGS stored procedure:

Field	Size	Description
RTCODE	Integer	The return code. A return code of 4 is a warning. A return code higher than 4 is an error.

CMSGS Example

DB2 connect call to CMSGS

call repowner.CMSGS('?', 'XIO',27,'ELEMENT_NAME, DBXT, 0?',?)

The output will be as follows:

MSG_QUAL: NULL

MSG_ID: NULL

REASON: Error: ELEMENT_NAME, DBXT, 0already exists in the repository.

RTCODE: 0

CRTXT80

CRTXT80 is a stored procedure that processes text from tables DBX_TEXT_1 through DBX_TEXT_5.

Input

The following table describes the fields processed by the CRTXT80 stored procedure:

Field	Size	Description	Parameter Number
Action	1 char	The text action that will be implemented, as follows: I-Insert; inserts text into the	1

Field	Size	Description	Parameter Number
		specified text table for a given entity ID **Note:** An error occurs if text already exists for the entity ID D-Delete; deletes text from the specified text table for a given entity ID S-Select; selects the rows from the specified text table for a given entity ID R-Replace; deletes the existing text and inserts the new text for the specified text table for a given entity ID U-Update; appends text to the specified text table for a given entity ID	

Field	Size	Description	Parameter Number
Action (cont.)	1 char	V-Move; moves text in the specified text table from the specified entity ID to the specified move ID. **Note:** An error occurs if text already exists for the move ID G-Merge; appends text in the specified text table from the specified entity ID to the specified move ID table. C-Copy; copies text in the specified text table from the specified entity id to the specified move id. **Note:** An error occurs if text already exists for the move ID A-Delete all; deletes text from all text tables for a given entity ID E-Move all; moves text in all text tables from the specified entity ID to the specified move ID **Note:** An error occurs if text already exists for the move ID in any text F - Merge all; appends text from all text	1

Field	Size	Description	Parameter Number
		tables from the specified entity ID to the specified move ID P - Copy all; copies text from all text tables from the specified entity id to the specified move ID **Note:** An error will occur if text already exists for the move id in any text table. Always set the indicator variable to 0.	
Text type	1 char	The text table suffix of the table that the text processing is intended for The Text type can be 1 - 5.	2
Entity type	Half word	The entity type of the object for which text is being processed	3
Entity id	Full word	The entity ID of the object for which text is being processed	4
Move id	Full word	The object ID of the object whose text is updated with the text from the object located in entity ID **Note:** Move ID is required if the action is V, E, G, F, C, or P.	5
Move type	Half word	The entity type of the object with the move ID **Note:** Move type is required if the action is V, E, G, F, C, or P.	6
Move_ userid	8 Varchar	The user ID that matches the audit_userid in the target text of a merge operation.	7
Move_tim estamp	26 varchar	The timestamp that matches the audit_timestamp in the target text of a merge operation. Used to check if the text can be acted on.	8

Input-Output

The following table describes the fields that serve as both input and output fields processed by the CRTXT80 stored procedure:

Field	Size	Description	Parameter Number
Ent_userid	8 varchar	The userid that matches the audit_userid Used to check if the text can be acted upon.	9
Ent_timestamp	26 varchar	The timestamp that matches the audit_timestamp Used to check if the text can be acted on.	10

Output

The following table describes the fields processed by the CRTXT80 stored procedure:

Field	Size	Description	Parameter Number
Return code	Full word	A return code other than zero indicates an error	11
Message qualifier	3 chars	The error message prefix	12
Message id	Full word	The message ID	13
Reason	80 chars varying	The short message text	14

Notes:

- The stored procedure is defined as General with NULLS
- An indicator variable must be set for all input and output parameters
- Set all output variable indicators to -1
- Set any unused input variable indicators to -1

  Note: Setting output variable indicators and unused input variable indicators to -1 will improve performance, as these fields will not be transmitted to the stored procedure.
- Set all used input variables indicators to 0
- Setting the entity userid and the entity timestamp variable indicators to -1, bypasses the concurrent update check, allowing the user to update the same text at the same time

The Global Temporary Table

A global temporary table called DBX_TEMP_TEXT was created at installation time. The table is based on DBX_TEXT_1. The temporary table contains the columns ENT_TYPE, ENT_ID, SEQ_NUM, and TEXT.

Prior to calling CRTXT80, you must populate this table with the text to process if the Action is I, U or R.

The calling program must delete from the temporary table or perform a commit or rollback prior to populating the temporary table to hold new text.

WARNING! Care must be taken when performing commits. A commit automatically deletes the data from the temporary table.

A result set is returned to the calling program if the Action is S. The Result set cursor is named TEXT_CUR. The columns returned are SEQ_NUM and TEXT.

The stored procedure is defined as COMMIT ON RETURN NO. The calling program must perform the commit or rollback.

CRTXT80 Example

Text processing:

```
/* ASSIGN THE INPUT PARAMETERS FOR THE TEXT PROCESSING STORED  */
/* PROCEDURE.                                                  */
TEXT_ACTION = 'S';    /* ACTION SELECT     */
PROCESS_TEXT = 1;     /* TEXT_TABLE SUFFIX */
ENT_TYPE  = 214;
ENT_ID    = 20;
/* SET UP INDICATORS */
ACTIONIND   = 0;
TEXTTYPEIND = 0;
ENTTYPEIND  = 0;
ENTIDIND    = 0;
MOVEIDIND   = -1;
MOVETYPEIND = -1;
MOVEUSERIND = -1;
MOVETMPIND  = -1;
ENTUSERIND  =  0;
ENTTMPIND   =  0;
RTCODEIND   = -1;
MSGQUALIND  = -1;
MSGIDIND    = -1;
REASONIND   = -1;
/* CALL TEXT PROCESSING STORED PROCEDURE */

EXEC SQL
  CALL CRTXT80(
     :TEXT_ACTION                      INDICATOR  :ACTIONIND,
     :PROCESS_TEXT                     INDICATOR  :TEXTTYPEIND,
     :ENT_TYPE                 INDICATOR  :ENTTYPEIND,
     :ENT_ID                           INDICATOR  :ENTIDIND,
     :MOVE_ID                          INDICATOR  :MOVEIDIND,
     :MOVE_TYPE                INDICATOR  :MOVETYPEIND,
      MOVE_USER                INDICATOR   MOVEUSRIND,
      MOVE_TIMES               INDICATOR   MOVETMPIND,
      ENT_USER                 INDICATOR   ENTUSERIND,
      ENT_TIMES                INDICATOR   ENTTMPIND,
     :RET_CODE                 INDICATOR  :RTCODEIND,
     :MSG_QUAL                 INDICATOR  :MSGQUALIND,
     :MSG_ID                           INDICATOR  :MSGIDIND,
     :REASON                           INDICATOR  :REASONIND) ;
IF SQLCODE ^= 0 & SQLCODE ^= 466
  THEN RETURN_CODE = SQLCODE;
  ELSE DO;
     IF RET_CODE ^= 0
       THEN RETURN_CODE = RET_CODE;
       ELSE RETURN_CODE = 0;
  END;
```

DBXENTID

The DBXENTID stored procedure returns the entity ID of an instance of a repository entity, relationship or association.

Input

The following table describes the input fields processed by the DBXENTID stored procedure:

Field	Size	Description	Parameter Number
Entity type	Half word	The entity type of the object being searched.	1
Name	245 characters varying	The value in the name attribute of the Entity, relationship or set being searched. If the entity_type represents an association then the name refers to the value in the source object's name attribute.	2
Status	8 characters	The value in the status attribute of the Entity, relationship or set being searched. If the entity_type represents an association then the status refers to the value in the source object's status attribute	3
Version	Half word	The value in the version attribute of the Entity, relationship or set being searched. If the entity_type represents an association then the version refers to the value in the source object's version attribute.	4
TargetName	245 characters varying	The value in the target object's name attribute if the entity type being searched is an association. Otherwise this input value is ignored.	5
TargetStatus	8 characters	The value in the target object's status attribute if the entity type being searched is an association. Otherwise this input value is ignored.	6
TargetVersion	Half word	The value in the target object's version attribute if the entity type being searched is an association.	7

Field	Size	Description	Parameter Number
		Otherwise this input value is ignored.	

Output

The following table describes the Output fields produced by the DBXENTID stored procedure:

Field	Size	Description
ENT_ID	Integer	The entity ID.
MSG QUAL	Char(3)	The message prefix.
MSG ID	Integer	The message ID.
RTCODE	Integer	The return code. A return code of 4 is a warning. A return code higher than 4 is an error.

DBXENTID Example

```
DB2 connect call to DBXENTID for an instance of an entity, relationship, set and
association.
call ARZR720.DBXENTID(25135,'TEXT2','DBXT', 0, ' ',' ',0,?,?,?,?,?)  --- Entity
call ARZR720.DBXENTID(26010,'.ALDD3333','TFS814', 2, ' ',' ',0,?,?,?,?,?)  ---
Relationship
call ARZR720.DBXENTID(102,'QATOOL.RENTAL_FEE','DBXT', 0, ' ',' ',0,?,?,?,?,?)  ---
Set
call ARZR720.DBXENTID(25130,'CCTB','DBXT', 0, 'DBMS OBJECTS','DBXT',0,?,?,?,?,?)
--- Association
```

DBXIA

DBXIA is a stored procedure that performs impact analysis.

To use the DBXIA stored procedure

1. The Entity ID is passed to the stored procedure.
2. The procedure returns a result set containing a list of objects that are related to the input entity ID.

Differences between the Mainframe XREF function and the Impact Analysis Stored procedure

- The XREF function does not display objects that are not in the current dialog. The new stored procedure will display objects that are not in the current dialog as long as the user has the security to see that object.
- The current program calls the IO module DBXVLIO to get the object. It then calls DBXMOVE to retrieve the name status and version from the buffer. The new program will instead call DBXNSVI to get the name status and version (Performance improvement)
- If an association is being display the Rel name and Rel status will be blank. The Rel Version will contain a -1.
- Because the stored procedure works outside the bounds of a dialog a user that does not have the authority to see an object will not see the object. We will return a warning message as output parameters of the stored procedure. The output parameter MSG_ QUAL will contain "EDT' The MSG_ID will be 57 The return code will be 4.

Example

a. Impact analysis is performed on a specific row in object Table and the user is not authorized to see the object Column

   No rows will be returned for Column. If you can't see a relationship or association you can't see any object it is related to.

b. Impact analysis is performed on a specific row in object Table and the user is not authorized to see the object Element

   No rows will be returned for any relationship to Element. If you can't see an object you can't see any relationships or associations to it.

c. Impact analysis is performed on a specific row in object Table and the user is not authorized to see some rows in the object Column

   No row will be returned if the user is not authorized to see the status of the Column

d. Impact analysis is performed on a specific row in object Table and the user is not authorized to see some rows in the object Element

   No row will be returned if the user is not authorized to see the status of the Element

Input

The following table describes the input fields processed by the DBXIA stored procedure:

Field	Size	Description	Parameter Number
Entity type	Integer	The entity ID of the object for which.	1

Output

The following table describes the output fields produced by the DBXIA stored procedure:

Field	Size	Description
MSG QUAL	Char(3)	The message prefix.
MSG ID	Integer)	The message ID.
RTCODE	Integer	The return code. A return code of 4 is a warning. A return code higher than 4 is an error.
REASON	Varchar(80)	The short message text.

Note: Be aware of the following:

- The stored procedure is defined as General with NULLS.
- An indicator variable must be set for all output parameters.
- All output variables indicators must be set to -1.
- The cursors used in this routine are defined "with hold" and "with return" to hold and return the result set back to the caller.
- A result set is returned to the calling program as a result of opening the cursor for the Select statement. The Result set cursor is named XREF_CSR.
- The stored procedure is defined as COMMIT ON RETURN NO.

Result set consists of following columns:

ENT ID ENT TYPE ENT TYPE NAME	Entity id, entity type and entity type name of related Entity or Relationship
ASSOC_FLAG	Y - if relates thru association, N - if relates thru relationship
IMPACT_TYPE	U - related object is a target, W - related object is a source
ENT_NAME ENT_STATUS ENT_VERSION	Name, status and version of related object.
REL_NAME REL_STATUS REL_VERSION	Name, status and version of connected object
REL_ENT_ID REL_TYPE REL_TYPE_NAME	Entity id, entity type and entity type name of connected Relationship or association.

DBXIA Example

```
Return the results of an impact analysis for a table with
entity id = 9464
DB2 connect call to DBXIA
CALL repowner.DBXIA(9464,? ,?, ?, ?);
```

DBXIOD

DBXIOD is a stored procedure that deletes objects from the repository.

It works this way:

1. The entity type and entity ID are passed to the procedure.
2. The object along with all the associated sources and targets will be deleted.
3. A count of the number of objects deleted is returned to the calling program.

Notes:

- The delete will not be allowed, if the object or any of its sources and targets is locked in update mode.
- When an object is deleted, rows with that entity ID are deleted from the object table, cross reference table, workstation cross reference table, and text tables. A row is written to the lock activity table if the object contained a select lock.

Input

The following table describes the input fields processed by the DBXIOD stored procedure:

Field	Size	Description	Parameter Number
Entity type	Half word	The entity type of the object being deleted.	1
Entity id	Full word	The entity id of the object being deleted	2

Output

The following table describes the output fields produced by the DBXIOD stored procedure:

Field	Size	Description	Parameter Number
Delete count	Full word	The number of objects deleted	3
Return code	Full word	A return code other than zero indicates an error	4

Field	Size	Description	Parameter Number
Message qualifier	3 chars	The error message prefix	5
Message id	Full word	The message ID	6
Reason	80 chars varying	The short message text	7

Notes:

- The stored procedure is defined as General with NULLS.
- An indicator variable must be set for all output parameters.
- Set all output variable indicators to -1.

  Setting output variable indicators improves performance, as these fields are not transmitted to the stored procedure.

The stored procedure is defined as COMMIT ON RETURN NO. The calling program must perform the commit or rollback.

DBXIOI

DBXIOI is a stored procedure that inserts objects in the repository. This stored procedure checks the following before the actual insert process:

- Existence and validity of the source and target entities
- Status of the object, source, and target entities

The stored procedure first finds the next available entity ID in the cross-reference table, DBX_XREF, and inserts a row in that table. Then, the actual insert of the object in the repository is done.

The stored procedure checks for the following cases:

- Duplicate entity
- Duplicate target name
- Duplicate association
- Duplicate relationship
- Any policy restriction

- Maximum number of source rows for a particular target in the cross-reference table, DBX_XREF, exceed the allowable maximum
- Maximum number of target rows for a particular source in the cross-reference table, DBX_XREF, exceed the allowable maximum

The Insert process stops and error message are issued if any of the previous checks fail.

Notes:

- New columns are inserted in the repository based on the attributes in the map definition structure and by the values in the block data area. The valid data types are CHAR, VARCHAR, DATE, TIME, TIMESTAMP, PACKED DECIMAL, INTEGER, and SMALLINT. An error message will be issued for invalid data types.
- The columns with character (C) data type and with either the value of blank in the data block or the audit flag of N is inserted with the value in the USER special register of DB2.
- The columns with date, time, and timestamp data types are inserted with the current values in the corresponding DB2 special registers.
- A new row is added to the workstation cross-reference table, DBX_XREF, if the workstation id is passed to the stored procedure.

Parameters

The following table describes the parameters used by the DBXIOI stored procedure:

Field	Size	Description	Parameter Number
Map definition (input)	13200 chars varying	The map definition refers to the structure that holds the information to be used to insert the columns of data in the repository. For every column inserted, there are 44 bytes of information in the map definition structure such as column name, column type, offset and length of the value in the data block that is inserted in the repository. The fields in the map definition structure are shown below.	1
Block data (input)	4044 chars	The block of data that its values are used to insert the columns in the	2

Field	Size	Description	Parameter Number
	varying	repository.	
Entity id (output)	Full word	The entity ID of the object being inserted.	3
Entity type (input)	Half word	The entity type of the object being inserted.	4
Map id (input)	Full word	The entity map ID of the object being inserted.	5
Source id (inout)	Full word	The source entity ID.	6
Target id (inout)	Full word	The target entity ID.	7
S_t_select (input)	1 chars	This flag is optional. **Default:** N if not passed	8
Validate (inout)	1 char	The validation process flag. N means no validation is done and anything other than N indicates the validation will be performed. Validation checks for the migration path, statuses, and policy restrictions.	9
Migrate (input)	1 char	The migration flag. Anything other than M along with the other flags causes the validation of statuses and checking the maximum allowable rows for target and source entities in the cross-reference table, DBX_XREF.	10
Save audit (input)	1 char	The audit flag. Y indicates all audit attributes with the attr_code of "C" in the map definition will be set to the values from the SQL statement or data file. Otherwise, audit attributes are set to the current values in the corresponding DB2 special registers.	11
Workstation id (input)	50 chars	The workstation ID is used for processing the workstation group. This field is optional and if not passed, the workstation information is not added to the workstation cross-reference table,	12

Field	Size	Description	Parameter Number
		DBX_WKSN_XREF.	
Workstation data (input)	254 chars	(Optional). The data associated with the workstation ID (or blank if not passed) that is added to the workstation cross-reference table, DBX_WKSN_XREF.	13
Return code (output)	Full word	A return code other than zero indicates an error.	14
Message qualifier (output)	3 chars	The error message prefix.	15
Message id (output)	Full word	The message ID.	16
Reason (output)	80 chars varying	The short message text.	17

Notes:

- The stored procedure is defined as General with NULLS
- An indicator variable must be set for all output parameters
- All output variable indicators must be set to -1
- The indicator variables must be set to -1 for all optional variables that have not been passed to the stored procedure.

  The optional variables are S_t_select, Workstation ID, and Workstation data.
- Setting output variable indicators will improve performance, as these fields will not be transmitted to the stored procedure
- The stored procedure is defined as COMMIT ON RETURN NO. The calling program must perform the commit or rollback

The name, type, and the length of the fields in the Map definition structure that is used by the Insert stored procedure are:

Field	Size	Description
Column_name	18 chars	The name of the column to be inserted
Attr_code	1 char	(Attribute code)
Unused	1 char	
Type	1 char	The type of the inserted data

Field	Size	Description
Source_target	1 char	
Offset	half word	The offset of the insert value in the block data
Byte_length	half word	The length of the insert value in the block data
Unused	18 chars	

DBXIOI Example

```
Insert entity:

/* ASSIGN ALL THE PARAMETERS REQUIRED FOR THE INSERT ENTITY    */ /* STORED PROCEDURE
SUCH AS THE EXAMPLE BELOW THAT SHOWS         */
/* ASSIGNMENT TO FOUR OF THE PARAMETERS.                          */
ENT_TYPE   = 204;
MIGRATE    = 'M';
VALIDATE   = 'B';
SAVE_AUDIT = 'Y';

/* SET UP INDICATORS */
MAPDEFIND   =  0;
BLKDATAIND  =  0;
ENTIDIND    = -1;
ENTTYPEIND  =  0;
MAPIDIND    =  0;
SOURCEIDIND =  0;
TARGETIDIND =  0;
VALIDATEIND =  0;
MIGRATEIND  =  0;
SAVAUDITIND =  0;
IF S_T_SELECT = 'Y'
  THEN STSELECTIND =  0;
  ELSE STSELECTIND = -1;
IF WORKSTATION_ID = ' '
  THEN WRKSTNIDIND = -1;
  ELSE WRKSTNIDIND =  0;
IF WORKSTATION_DATA = ' '
  THEN WRKSTNDAIND = -1;
  ELSE WRKSTNDAIND =  0;
RTCODEIND   = -1;
MSGQUALIND  = -1;
MSGIDIND    = -1;
REASONIND   = -1;
```

```
/* CALL INSERT ENTITY STORED PROCEDURE */
EXEC SQL
  CALL DBXIOI(
     :MAPDEF                    INDICATOR  :MAPDEFIND,
     :BLOCK.DATA                INDICATOR  :BLKDATAIND,
     :ENT_ID                    INDICATOR  :ENTIDIND,
     :ENT_TYPE                  INDICATOR  :ENTTYPEIND,
     :MAP_ID                           INDICATOR  :MAPIDIND,
     :SOURCE_ID                 INDICATOR  :SOURCEIDIND,
     :TARGET_ID                 INDICATOR  :TARGETIDIND,
     :VALIDATE                  INDICATOR  :VALIDATEIND,
     :MIGRATE                   INDICATOR  :MIGRATEIND,
     :SAVE_AUDIT                INDICATOR  :SAVAUDITIND,
     :S_T_SELECT                INDICATOR  :STSELECTIND,
     :WORKSTATION_ID            INDICATOR  :WRKSTNIDIND,
     :WORKSTATION_DATA          INDICATOR  :WRKSTNDAIND,
     :RET_CODE                  INDICATOR  :RTCODEIND,
     :MSG_QUAL                  INDICATOR  :MSGQUALIND,
     :MSG_ID                    INDICATOR  :MSGIDIND,
     :REASON                    INDICATOR  :REASONIND);

IF SQLCODE ^= 0
  THEN RETURN_CODE = SQLCODE;
  ELSE DO;
     IF RET_CODE ^= 0
       THEN RETURN_CODE = RET_CODE;
       ELSE RETURN_CODE = 0;
  END;
```

DBXIOM

DBXIOM is a stored procedure that merges objects in the repository. It works this way:

1. The *from* and *to* merge entity IDs and entity types are passed to the procedure. The rows with the source and target IDs matching the *from* entity ID are replaced with the entity ID being merged *to* in the DBX_XREF table.
2. The DBXIOM stored procedure then merge the related text data in all the text tables by calling the text processing stored procedure, CRTXT80.

3. The next step in the DBXIOM stored procedure is to insert the new rows to the workstation table, DBX_WKSN_XREF, with information about the merges.
4. The *from* merge entity ID and all its relations are then deleted by calling the DBXIOD stored procedure.

Note: The merge process fails if trying to merge an association.

Input

The following table describes the input fields processed by the DBXIOM stored procedure:

Field	Size	Description	Parameter Number
Lock id	Full word	The entity ID of the object being merged from.	1
Lock type	Half word	The entity type of the object being merged from.	2
Entity id	Full word	The entity ID of the object being merged to.	3
Entity type	Half word	The entity type of the object being merged to.	4

Output

The following table describes the output fields produced by the DBXIOM stored procedure:

Field	Size	Description	Parameter Number
Return code	Full word	A return code other than zero indicates an error	5
Message qualifier	3 chars	The error message prefix	6
Message id	Full word	The message ID	7
Reason	80 chars varying	The short message text	8

DBXIOM Example

```
Merge entity:

/* ASSIGN THE INPUT PARAMETERS FOR THE MERGE ENTITY STORED       */

/* PROCEDURE.                                                    */
LOCK_ID   = 10;
LOCK_TYPE = 204;
ENT_ID    = 20;
ENT_TYPE  = 214;

/* SET UP INDICATORS */
LOCKIDIND   = 0;
LOCKTYPEIND = 0;
ENTIDIND    = 0;
ENTTYPEIND  = 0;
RTCODEIND   = -1;
MSGQUALIND  = -1;
MSGIDIND    = -1;
REASONIND   = -1;

/* CALL MERGE STORED PROCEDURE */
EXEC SQL
  CALL DBXIOM(
     :LOCK_ID                  INDICATOR  :LOCKIDIND,
     :LOCK_TYPE                 INDICATOR  :LOCKTYPEIND,
     :ENT_ID                    INDICATOR  :ENTIDIND,
     :ENT_TYPE           INDICATOR  :ENTTYPEIND,
     :RET_CODE                  INDICATOR  :RTCODEIND,
     :MSG_QUAL                  INDICATOR  :MSGQUALIND,
     :MSG_ID                    INDICATOR  :MSGIDIND,
     :REASON                    INDICATOR  :REASONIND) ;

IF SQLCODE ^= 0
  THEN RETURN_CODE = SQLCODE;
  ELSE DO;
     IF RET_CODE ^= 0
       THEN RETURN_CODE = RET_CODE;
       ELSE RETURN_CODE = 0;
  END;
```

DBXIOR

DBXIOR is a stored procedure that replaces objects in the repository by updating the target ID of the object being replaced to the entity ID of the replacing object. It works this way:

1. The existing and the replacing entity IDs are passed to the procedure. The rows with the target ID matching the existing entity ID are replaced with the replacing entity ID in the DBX_XREF table.
2. The replaced entity ID and all its relations are then deleted by calling the DBXIOD stored procedure.

The replace process fails if trying to replace an association.

Input

The following table describes the input fields processed by the DBXIOR stored procedure:

Field	Size	Description	Parameter Number
Entity ID	Full word	The entity ID of the object being replaced.	1
Entity type	Half word	The entity type of the object being replaced.	2
Lock id	Full word	The replacing entity ID	3

Output

The following table describes the output fields produced by the DBXIOR stored procedure:

Field	Size	Description	Parameter Number
Return code	Full word	A return code other than zero indicates an error	4
Message qualifier	3 chars	The error message prefix	5
Message id	Full word	The message ID	6
Reason	80 chars	The short message text	7

Field	Size	Description	Parameter Number
	varying		

Notes:

- The stored procedure is defined as General with NULLS.
- An indicator variable must be set for all output parameters.
- Set all Output variable indicators to -1.
- Setting output variable indicators will improve performance, as these fields will not be transmitted to the stored procedure.
- The stored procedure is defined as COMMIT ON RETURN NO. The calling program must perform the commit or rollback.

DBXIOR Example

```
Replace entity:

/* ASSIGN THE INPUT PARAMETERS FOR THE REPLACE ENTITY STORED     */
/* PROCEDURE.                                                    */


ENT_ID    = 20;
ENT_TYPE  = 214;
LOCK_ID   = 10;


/* SET UP INDICATORS */

ENTIDIND   = 0;
ENTTYPEIND = 0;
LOCKIDIND  = 0;
RTCODEIND  = -1;
MSGQUALIND = -1;
MSGIDIND   = -1;
REASONIND  = -1;
```

```
/* CALL REPLACE STORED PROCEDURE */

EXEC SQL
  CALL DBXIOR(
     :ENT_ID                    INDICATOR  :ENTIDIND,
     :ENT_TYPE           INDICATOR  :ENTTYPEIND,
     :LOCK_ID                   INDICATOR  :LOCKIDIND,
     :RET_CODE                  INDICATOR  :RTCODEIND,
     :MSG_QUAL                  INDICATOR  :MSGQUALIND,
     :MSG_ID                    INDICATOR  :MSGIDIND,
     :REASON                    INDICATOR  :REASONIND) ;

IF SQLCODE ^= 0
  THEN RETURN_CODE = SQLCODE;
  ELSE DO;
     IF RET_CODE ^= 0
       THEN RETURN_CODE = RET_CODE;
       ELSE RETURN_CODE = 0;
  END;
```

DBXIOS

DBXIOS is a Select stored procedure. It returns the result set of a single select or multiple fetch of a table back to the caller for further processing.

Be aware of the following:

- The same routine is invoked by 15 different stored procedures (dbxios01 thru dbxios15) to handle multiple tables concurrently.
- The cursors used in this routine are defined "with hold" and "with return" to hold and return the result set back to the caller. Since this routine is invoked by 15 stored procedures, it is defined with 15 cursors each used by one of the stored procedures based on the cursor number parameter passed to it.
- The stored procedure builds the Select statement dynamically based on the information passed to it. The Where clause of the Select statement is built with parameter markers. The statement is then prepared and opened to get the result set.

Input

The following table describes the input fields processed by the DBXIOS Select stored procedure:

Field	Size	Description	Parameter Number
Action	1 char	The action code can be either Select (S) for single select or Open (O) for multiple fetch.	1
Map definition	13200 chars varying	The map definition refers to the structure that holds the information used to select/fetch the columns of data from a table in the repository. For every column selected, there are 44 bytes of information in the map definition structure such as column name, and column type. The fields in the map area structure are shown below.	2
Block data	4044 chars varying	The block of data used to build the Where clause for the Select statement.	3
Entity id	Full word	The entity identification of the selected object. It is used in the Where clause of the Select statement if it is not zero.	4
Entity type	Half word	The entity type of the selected object. It is used in the Where clause of the Select statement.	5
Source id	Full word	The source ID of the selected object. It is used in the Where clause of the Select statement if it is not zero.	6
Target id	Full word	The target ID of the selected object. It is used in the Where clause of the Select statement if it is not zero.	7
Lock id	Full word	The lock ID of the selected object. It is used in the Where clause of the Select statement if it is not zero.	8
TSO logon id	8 chars	The TSO logon ID of the selected object. It is used in the Where clause of the Select statement if it is	9

Field	Size	Description	Parameter Number
		not blank.	
Timestamp	26 chars	The timestamp of the selected object. It is used in the Where clause of the Select statement if it is not blank.	10
Map id	Full word	The map IDs used to get the names of the entity, source, target, and domain tables.	11
Cursor number	Half word	The cursor number, used to retrieve rows and build the result sets. It can be 1 through 15. The number is set to 1 if it is outside the range.	12
Domain select	1 char	(Optional). The domain of the Select statement. It determines the column and table names that need to be included in the Select statement. **Default:** N if not passed	13
Where data	32704 chars varying	Additional where criteria that is added to the Where clause of the Select statement.	14
Order by	250 chars	The list of the columns to use in the Order by clause of the Select statement.	15

Output

The following table describes the output fields produced by the DBXIOS Select stored procedure:

Field	Size	Description	Parameter Number
Return code	Full word	A return code other than zero indicates an error	16
Message qualifier	3 chars	The error message prefix	17
Message id	Full word	The message ID	18
Reason	80 chars	The short message text	19

Field	Size	Description	Parameter Number
Return code	Full word	A return code other than zero indicates an error	16
	varying		

Be aware of the following:

- The stored procedure is defined as General with NULLS.
- An indicator variable must be set for all output parameters.
- All output variables indicators must be set to -1.
- The indicator variables must be set to -1 for all optional variables that were not passed to the stored procedure. Domain select is the optional variable.

  Note: Setting output variable indicators improves performance, as these fields are not transmitted to the stored procedure.
- A result set is returned to the calling program as a result of opening the cursor for the Select statement. The Result set cursor is named SEL_CSR.
- The stored procedure is defined as COMMIT ON RETURN NO. The calling program must perform the commit or rollback.

The name, type, and the length of the fields in the Map definition structure are used by the Select stored procedure are:

Field	Size	Description
Column_name	18 chars	The name of the column to be selected
Attr_code	1 char	Attribute code
Domain_use	1 char	
Type	1 char	The type of the selected data
Source_target	1 char	
Offset	half word	The offset of the data area in the block
Byte_length	half word	The length of the data to be selected
Oreder_num	half word	The order by column if non-zero
Oreder_ord	1 char	Specifies the ascending or descending order

Field	Size	Description
Unused	15 chars	

DBXIOS Example

```
Select entity:

IDENTIFICATION DIVISION.
   PROGRAM-ID.  IOSSAMP.
  ****************************************************************
  *
  *    A PROGRAM THAT TESTS THE SELECT STORED PROCEDURE
  *
  ****************************************************************
  ****************************************************************
   EJECT
  ****************************************************************
   ENVIRONMENT DIVISION.
  ****************************************************************
   INPUT-OUTPUT SECTION.
   EJECT
  ****************************************************************
```

```
 DATA DIVISION.
*****************************************************************
 WORKING-STORAGE SECTION.
*****************************************************************
 01 MISC-FIELDS.
     05 DISPLAY-SQLCODE                  PIC -999.
 01  EOF-SW                       PIC X VALUE 'N'.
     88  NO-MORE-ROWS             VALUE 'Y'.
 01 INPUT-FIELDS.
     05 ACTION                           PIC X(01).
     05 MAPDEF.
         49 MAPDEFLEN                    PIC S9(4) USAGE COMP.
         49 MAPDEFTXT                    PIC X(13200).
     05 BLKDATA.
         49 BLKDATALEN                   PIC S9(4) USAGE COMP.
         49 BLKDATATXT                   PIC X(4044).
     05 ENT-ID                           PIC S9(9) USAGE COMP.
     05 ENT-TYPE                         PIC S9(4) USAGE COMP.
     05 SOURCE-ID                        PIC S9(9) USAGE COMP.
     05 TARGET-ID                        PIC S9(9) USAGE COMP.
     05 LOCK-ID                          PIC S9(9) USAGE COMP.
     05 TSO-LOGON                        PIC X(08).
     05 TIMESTAMP                        PIC X(26).
     05 MAP-ID                           PIC S9(9) USAGE COMP.
     05 CURSOR-NUM                       PIC S9(4) USAGE COMP.
     05 DOMAIN-SELECT                    PIC X(01).
     05 WHERE-DATA.
         49 WHERE-DATALEN                PIC S9(4) USAGE COMP.
         49 WHERE-DATATXT                PIC X(32704).
     05 ORDER-BY                         PIC X(250).
 01 OUTPUT-FIELDS.
     05 RET-CODE                         PIC S9(9) USAGE COMP.
     05 MSG-QUAL                         PIC X(03).
     05 MSG-ID                           PIC S9(9) USAGE COMP.
     05 REASON.
         49 REASONLEN                    PIC S9(4) USAGE COMP.
         49 REASONTXT                    PIC X(80).
     05 UNUSED                           PIC X(15).
 01 OUT-FLDS.
     05 OUT-ENT-TYPE                     PIC S9(4) COMP.
     05 OUT-ENT-ID                       PIC S9(9) COMP.
     05 OUT-SOURCE-ID                    PIC S9(9) COMP.
     05 OUT-TARGET-ID                    PIC S9(9) COMP.
     05 OUT-LOCK-ID                      PIC S9(9) COMP.
     05 OUT-USER                         PIC X(8).
     05 OUT-TIMESTAMP                    PIC X(26).
 01 BLOCK-REC.
     05 PGM-NAME                         PIC X(32).
     05 PGM-STATUS                       PIC X(08).
```

```
    05 PGM-VERSION                      PIC X(06).
    05 PGM-LANGUAGE                     PIC X(01).
01 MAP-DEF.
   03  ROW-DEF OCCURS 4   TIMES.
     05 MAP-COL-NAME     PIC X(18).
     05 MAP-ATTR-CODE    PIC X.
     05 MAP-DOMAIN-USE   PIC X.
     05 MAP-TYPE         PIC X.
     05 MAP-SRC-TARG     PIC X.
     05 MAP-OFFSET       PIC S9(4) COMP.
     05 MAP-BYTELEN      PIC S9(4) COMP.
     05 MAP-ORDERNUM     PIC S9(4) COMP.
     05 MAP-ORD-ASCDESC  PIC X.
     05 FILLER           PIC X(15).


*INCLUDE INDICATOR VARIABLES FOR EVERY FIELD
*PASSED TO AND FROM THE STORED PROCEDURE.
01 INDICATORS.
  05 ACTIONIND                        PIC S9(4) COMP.
  05 MAPDEFIND                        PIC S9(4) COMP.
  05 BLKDATAIND                       PIC S9(4) COMP.
  05 ENTIDIND                         PIC S9(4) COMP.
  05 ENTTYPEIND                       PIC S9(4) COMP.
  05 SOURCEIDIND                      PIC S9(4) COMP.
  05 TARGETIDIND                      PIC S9(4) COMP.
  05 BLKLOCKIND                       PIC S9(4) COMP.
  05 TSOIDIND                         PIC S9(4) COMP.
  05 BLKTIMIND                        PIC S9(4) COMP.
  05 MAPIDIND                         PIC S9(4) COMP.
  05 CURSORIND                        PIC S9(4) COMP.
  05 DOMAINSELIND                     PIC S9(4) COMP.
  05 WHEREDATAIND                     PIC S9(4) COMP.
  05 ORDERBYIND                       PIC S9(4) COMP.
  05 RTCODEIND                        PIC S9(4) COMP.
  05 MSGQUALIND                       PIC S9(4) COMP.
  05 MSGIDIND                         PIC S9(4) COMP.
  05 REASONIND                        PIC S9(4) COMP.
```

```
* INCLUDE A RESULT SET LOCATOR FOR THE RESULT SET
* THAT IS RETURNED.
 01 LOC                          USAGE SQL TYPE IS
                                 RESULT-SET-LOCATOR VARYING.
* INCLUDE SQLCA
     EXEC SQL INCLUDE SQLCA END-EXEC.
 PROCEDURE DIVISION.
* ASSIGNMENT TO FOUR OF THE PARAMETERS.
      INITIALIZE  INPUT-FIELDS.
      INITIALIZE  BLOCK-REC.
      INITIALIZE  MAP-DEF.
*   IF ACTION IS SET TO S THEN THE QUERY WILL BE
*     OPTIMIZED FOR 1 ROW.
*   IF ACTION SET TO O THE OPTIMIZE KEYWORD IS NOT USED.
      MOVE 'O'                     TO ACTION
      MOVE 151                     TO ENT-TYPE
*   ONLY NEED MAP_ID TO GET THE NAME OF THE
*   TABLES BEING PROCESSED. THE DBXIOS PROGRAM DOES
*   NOT USE THE MAP TO GET THE ATTRIBUTES.  THE
*   ATTRIBUTES ARE DEFINED IN THE MAP-DEF RECORD.
      MOVE 151                     TO MAP-ID
*     MOVE 58855                   TO ENT-ID
      MOVE  1                      TO CURSOR-NUM.
*  SET UP MAP-DEF  FOR 4 COLUMNS PGM_NAME, STATUS, VERSION
*     AND LANGUAGE.
     MOVE 'PGM_NAME' TO MAP-COL-NAME(1).
     MOVE 'N' TO MAP-ATTR-CODE(1).
     MOVE 'C' TO MAP-TYPE(1).
     MOVE 1 TO MAP-OFFSET(1).
     MOVE 32 TO MAP-BYTELEN(1).
     MOVE 'STATUS' TO MAP-COL-NAME(2).
     MOVE 'S' TO MAP-ATTR-CODE(2).
     MOVE 'C' TO MAP-TYPE(2).
     MOVE 33 TO MAP-OFFSET(2).
     MOVE 8 TO MAP-BYTELEN(2).
     MOVE 'VERSION' TO MAP-COL-NAME(3).
     MOVE 'V' TO MAP-ATTR-CODE(3).
     MOVE 'C' TO MAP-TYPE(3).
     MOVE 41 TO MAP-OFFSET(3).
     MOVE 6 TO MAP-BYTELEN(3).
     MOVE 'LANGUAGE' TO MAP-COL-NAME(4).
     MOVE ' ' TO MAP-ATTR-CODE(4).
     MOVE 'C' TO MAP-TYPE(4).
     MOVE 47 TO MAP-OFFSET(4).
     MOVE 1 TO MAP-BYTELEN(4).
*  SET UP INPUT TO STORED PROCEDURE
*  LENGTH OF MAPDEF IS 44 BYTES PER OCCURANCE
     MOVE MAP-DEF TO MAPDEFTXT.
```

```
     MOVE 176 TO MAPDEFLEN.
*  LENGTH OF BLOCKDATA IS 4044
     MOVE SPACES TO BLKDATATXT.
     MOVE 4044 TO BLKDATALEN.
*  SET UP BLOCK DATA TO CONTAIN FIELDS TO BE USED IN THE
*  WHERE CLAUSE.  ONLY NON BLANK FIELDS WILL BE USED.
*  SET LANGUAGE TO P TO GET ALL PROGRAMS WHERE
*  THE LAGUAGE EQUALS P
     MOVE 'P' TO PGM-LANGUAGE.
     MOVE BLOCK-REC TO BLKDATATXT.
* SET UP INDICATORS */
     MOVE 0 TO  ACTIONIND
     MOVE 0 TO  BLKDATAIND
     MOVE -1 TO  ENTIDIND
     MOVE 0 TO  ENTTYPEIND
     MOVE 0  TO  SOURCEIDIND
     MOVE 0  TO  TARGETIDIND
     MOVE 0  TO  BLKLOCKIND
     MOVE 0 TO  TSOIDIND
     MOVE 0  TO  BLKTIMIND
     MOVE 0 TO  MAPIDIND
     MOVE 0 TO  CURSORIND
     MOVE -1 TO  DOMAINSELIND
     MOVE -1 TO  WHEREDATAIND
     MOVE -1 TO  ORDERBYIND
     MOVE 0 TO MAPDEFIND
     MOVE -1 TO RTCODEIND
     MOVE -1 TO MSGQUALIND
     MOVE -1 TO MSGIDIND
     MOVE -1 TO REASONIND
* CALL SELECT ENTITY STORED PROCEDURE */
      EXEC SQL
          CALL DBXIOS01(
          :ACTION                INDICATOR :ACTIONIND,
          :MAPDEF                INDICATOR :MAPDEFIND,
          :BLKDATA               INDICATOR :BLKDATAIND,
          :ENT-ID                INDICATOR :ENTIDIND,
          :ENT-TYPE              INDICATOR :ENTTYPEIND,
          :SOURCE-ID             INDICATOR :SOURCEIDIND,
          :TARGET-ID             INDICATOR :TARGETIDIND,
          :LOCK-ID               INDICATOR :BLKLOCKIND,
          :TSO-LOGON             INDICATOR :TSOIDIND,
          :TIMESTAMP             INDICATOR :BLKTIMIND,
          :MAP-ID                INDICATOR :MAPIDIND,
          :CURSOR-NUM            INDICATOR :CURSORIND,
          :DOMAIN-SELECT         INDICATOR :DOMAINSELIND,
          :WHERE-DATA            INDICATOR :WHEREDATAIND,
          :ORDER-BY              INDICATOR :ORDERBYIND,
          :RET-CODE              INDICATOR :RTCODEIND,
```

```
           :MSG-QUAL               INDICATOR :MSGQUALIND,
           :MSG-ID                 INDICATOR :MSGIDIND,
           :REASON                 INDICATOR :REASONIND)
       END-EXEC
      MOVE SQLCODE TO DISPLAY-SQLCODE.
      DISPLAY 'SQLCODE:  ' DISPLAY-SQLCODE.
      DISPLAY 'RETCODE:  ' RET-CODE.
      DISPLAY 'MSG-ID:   ' MSG-ID.
      DISPLAY 'MSG-QUAL: ' MSG-QUAL.
      DISPLAY 'REASON:   ' REASON.
      IF SQLCODE NOT = +466
         DISPLAY 'ERROR CALLING STORED PROCEDURE'
           DISPLAY 'SQLCODE:  ' DISPLAY-SQLCODE
           DISPLAY 'MSG-ID:   ' MSG-ID
           DISPLAY 'MSG-QUAL: ' MSG-QUAL
           DISPLAY 'REASON:   ' REASON
      ELSE
         IF RET-CODE NOT = 0
           DISPLAY 'SQLCODE:  ' DISPLAY-SQLCODE
           DISPLAY 'MSG-ID:   ' MSG-ID
           DISPLAY 'MSG-QUAL: ' MSG-QUAL
           DISPLAY 'REASON:   ' REASON
         ELSE
            PERFORM GET-RESULT-SET.
       GOBACK.
  GET-RESULT-SET.
      EXEC SQL
          ASSOCIATE LOCATORS (:LOC) WITH
             PROCEDURE DBXIOS01
      END-EXEC.
      EXEC SQL
          ALLOCATE NVSCURS CURSOR FOR RESULT SET :LOC
      END-EXEC.
      PERFORM GET-ROWS UNTIL NO-MORE-ROWS.
      EXEC SQL
          CLOSE  NSVCURS
      END-EXEC.
  GET-ROWS.
 *
 *  THE FIRST 7 FIELDS ARE REQUIRED.
 *
      EXEC SQL
         FETCH NVSCURS INTO
         :OUT-ENT-TYPE, :OUT-ENT-ID, :OUT-SOURCE-ID,
         :OUT-TARGET-ID, :OUT-LOCK-ID, :OUT-USER, :OUT-TIMESTAMP,
         :PGM-NAME, :PGM-STATUS, :PGM-VERSION, :PGM-LANGUAGE
      END-EXEC.
      IF SQLCODE = 100
       MOVE 'Y' TO EOF-SW
```

```
      END-IF.
      IF SQLCODE = 0
             DISPLAY OUT-ENT-TYPE
             DISPLAY OUT-ENT-ID
             DISPLAY OUT-USER
             DISPLAY OUT-TIMESTAMP
               DISPLAY 'PGM-NAME  ' PGM-NAME
               DISPLAY 'PGM-STATUS    ' PGM-STATUS
               DISPLAY 'PGM-VERSION   ' PGM-VERSION
      END-IF.
      IF SQLCODE < 0
        MOVE SQLCODE TO DISPLAY-SQLCODE
        DISPLAY DISPLAY-SQLCODE
        DISPLAY SQLERRM
        MOVE 'Y' TO EOF-SW
      END-IF.

END;
```

DBXIOU

DBXIOU is a stored procedure that updates objects in the repository.

It works this way:

1. The columns in the repository are updated based on the attributes in the map definition structure and by the values in the block data area. The valid data types are CHAR, VARCHAR, DATE, TIME, TIMESTAMP, PACKED DECIMAL, INTEGER, and SMALLINT. An error message is issued for invalid data types.

   The columns with character (C) data type and with either the value of blank in the data block or the audit flag of N are updated with the value in the USER special register of DB2. Likewise, the columns with date, time, and timestamp data types are updated with the current values in the corresponding DB2 special registers.

2. A new row is added to the workstation cross-reference table, DBX_XREF, if the workstation ID is passed to the stored procedure.

3. A row is added to the lock activity table if the object contains a select lock.

Notes:

- The stored procedure stops processing and issues an error message if any of the following checks fail:
  - Invalid migration path
  - Invalid object status
  - The object locked in the update mode
  - Duplicate entity found in the repository
  - Any policy restriction
- The Timestamp column is used for concurrence processing. It should not be changed after the Select and before the update.

Input

The following table describes the input fields processed by the DBXIOU stored procedure:

Field	Size	Description	Parameter Number
Map definition	13200 chars varying	The map definition refers to the structure that holds the information to be used to update the columns of data in the repository. For every column that is updated, there is 44 bytes of information in the map definition structure such as column name, column type, offset and length of the value in the data block that is updated in the repository. The fields in the map definition structure are shown below.	1
Block data	4044 chars varying	The block of data that its values are used to update the columns in the Repository.	2
Entity id	Full word	The entity id of the object being updated.	3
Entity type	Half word	The entity type of the object being updated.	4
Map id	Full word	The map id of the object being updated.	5
Dialog name	8 chars	The current dialog name. This field is optional and if not passed, the migration path will not be checked.	6

Field	Size	Description	Parameter Number
Validate	1 char	The validation process flag. N means no validation is done and anything other than N indicates the validation is performed. Validation will be checked for the migration path, statuses, and policy restrictions.	7
Timestamp	26 chars	The timestamp used for concurrence processing. This field is checked against the timestamp column in the cross-reference table, DBX_XREF, to make sure it has not been changed after select. It will then be updated with the current timestamp. An error will be returned if a blank timestamp is passed.	8
Save audit	1 char	The audit flag. Y indicates all audit attributes with the attr_code of C in the map definition are set to the values from the SQL statement or data file. Otherwise, audit attributes are set to the current values in the corresponding DB2 special registers.	9
Workstation ID	50 chars	The workstation ID is used for processing the workstation group. This field is optional and if not passed, the workstation information is not added to the workstation cross-reference table, DBX_WKSN_XREF.	10
Workstation data	254 chars	The data associated with the workstation ID (or blank if not passed) that is added to the workstation cross-reference table, DBX_WKSN_XREF. This field is optional.	11

Output

The following table describes the output fields produced by the DBXIOU stored procedure:

Field	Size	Description	Parameter Number
Return code	Full word	A return code other than zero indicates an error	12
Message qualifier	3 chars	The error message prefix	13
Message id	Full word	The message ID	14
Reason	80 chars varying	The short message text	15

Notes:

- The stored procedure is defined as General with NULLS.
- An indicator variable must be set for all output parameters.
- All Output variable indicators must be set to -1.

  The indicator variables must be set to -1 for all optional variables that were not passed to the stored procedure. The optional variables are Dialog name, Workstation ID, and Workstation data.

  Note: Setting output variable indicators improve performance, as these fields are not transmitted to the stored procedure.
- The stored procedure is defined as COMMIT ON RETURN NO. The calling program must perform the commit or rollback.
- The name, type, and the length of the fields in the Map definition structure that is used by the Update stored procedure are:

Field	Description	Size
Column_name	The name of the column to be updated	18 chars
Attr_code	Attribute code	1 char
Unused		1 char
Type	The type of the updated data	1 char
Source_target		1 char
Offset	The offset of the update value in the block data	half word

Field	Description	Size
Byte_length	The length of the update value in the block data	half word
Unused		17 chars
Mod	The description is modifiable	1 char

DBXIOU Example

```
Update entity:
/* ASSIGN ALL THE PARAMETERS REQUIRED FOR THE UPDATE ENTITY    */ /* STORED PROCEDURE
SUCH AS THE EXAMPLE BELOW THAT SHOWS         */
/* ASSIGNMENT TO FOUR OF THE PARAMETERS.                          */
ENT_ID   = 10;
ENT_TYPE = 204;
DIALOG   = 'DB2';
VALIDATE = 'N';

/* SET UP INDICATORS */
MAPDEFIND   = 0;
BLKDATAIND  = 0;
ENTIDIND    = 0;
ENTTYPEIND  = 0;
MAPIDIND    = 0;
IF DIALOG = ' '
  THEN DIALOGIND = -1;
  ELSE DIALOGIND = 0;
VALIDATEIND = 0;
TIMSTMPIND  = 0;
SAVAUDITIND = 0;
IF WORKSTATION_ID = ' '
  THEN WRKSTNIDIND = -1;
  ELSE WRKSTNIDIND = 0;
IF WORKSTATION_DATA = ' '
  THEN WRKSTNDAIND = -1;
  ELSE WRKSTNDAIND = 0;
RTCODEIND   = -1;
MSGQUALIND  = -1;
MSGIDIND    = -1;
REASONIND   = -1;
```

```
/* CALL UPDATE ENTITY STORED PROCEDURE */
EXEC SQL
  CALL DBXIOU(
     :MAPDEF                    INDICATOR  :MAPDEFIND,
     :BLOCK.DATA                INDICATOR  :BLKDATAIND,
     :ENT_ID                    INDICATOR  :ENTIDIND,
     :ENT_TYPE                  INDICATOR  :ENTTYPEIND,
     :MAP_ID                           INDICATOR  :MAPIDIND,
     :DIALOG                           INDICATOR  :DIALOGIND,
     :VALIDATE                  INDICATOR  :VALIDATEIND,
     :TIMESTAMP                        INDICATOR  :TIMSTMPIND,
     :SAVE_AUDIT                INDICATOR  :SAVAUDITIND,
     :WORKSTATION_ID            INDICATOR  :WRKSTNIDIND,
     :WORKSTATION_DATA          INDICATOR  :WRKSTNDAIND,
     :RET_CODE                  INDICATOR  :RTCODEIND,
     :MSG_QUAL                  INDICATOR  :MSGQUALIND,
     :MSG_ID                    INDICATOR  :MSGIDIND,
     :REASON                    INDICATOR  :REASONIND);

IF SQLCODE ^= 0
  THEN RETURN_CODE = SQLCODE;
  ELSE DO;
     IF RET_CODE ^= 0
       THEN RETURN_CODE = RET_CODE;
       ELSE RETURN_CODE = 0;
  END;
```

DBXNSVI

DBXNSVI is a stored procedure that returns the name, status and version of any Repository instance based on an ENT_ID.

Input

The following table describes the input fields processed by the DBXNSVI stored procedure:

Field	Size	Description	Parameter Number
Entity id	Full word	The entity ID of the object for which the name, status and version is being returned.	1

Output

The following table describes the output fields produced by the DBXNSVI Select stored procedure:

Field	Size	Description	Parameter Number
Entity type	Halfword	The entity type of the object being returned	2
Name	245 chars varying	The name of the instance being selected	3
Status	8 characters	The status of the instance being selected	4
Version	Halfword	The version number of the instance being selected	5
Return code	Full word	0 - Good return 4 - Row not found 8 - Error	6
Sqlcode	Full word	The sqlcode	7

Notes:

- The stored procedure is defined as General with NULLS.
- An indicator variable must be set for all input and output parameters.
- Set all output variable indicators to -1.
- Set any unused input variable indicators to -1

  Note: Setting output variable indicators and unused input variable indicators to -1 will improve performance, as these fields are not transmitted to the stored procedure.
- Set all used input variable indicators to 0.

DBXNSVI Example

DB2 connect call to DBXNSVI

call DBXREL30.DBXNSVI(226666, ?, ?, ?, ?, ?, ?)

DBXREL30

Is the repository qualifier

226666

Is a valid ENT_ID of a row in the repository

The results from DBXNSVI:

```
: 105

: CUSTOMER

: DBXT4

: 1

: 0

: NULL
```

DBXSEC

DBXSEC is a Security stored procedure and verifies the information such as function and command to be valid for a given TSO user id.

The stored procedure will stop processing and issue an error message during the security check process for the following conditions:

- No command passed.
- No TSO logon id passed.
- Invalid Command passed. It does not exist in the command validation table.
- Invalid Command passed. It does not exist in the command access table.
- Invalid Navigation command passed. It does not exist in the navigator definition table.
- Invalid Navigate command for the Continue function passed. It does not exist in the navigation execution table.

Parameters

The following table describes the parameters used by the DBXSEC stored procedure:

Field	Size	Description	Parameter Number
Window (input)	8 chars	The name of the window	1
Function (inout)	8 chars	The function of the requested command to be validated for security	2
Subfunction (inout)	8 chars	The sub function of the requested command to be validated for security	3
Command (inout)	8 chars	The requested command to be validated for security	4
Entity type (input)	Half word	The entity type	5
Status (input)	8 chars	The life cycle stage of the entity	6
TSO logon (input)	8 chars	The TSO user id checked by security procedure for allowing the requested action	7
Dialog (input)	8 chars	The dialog associated with the entity	8
Exit proc (output)	8 chars	The exit procedure name	9
Exit type (output)	1 char	The exit type, for example, C (clist), G (program)	10
Exit execution (output)	8 chars	The exit execution time, i.e. before or after	11
Return code (output)	Full word	A return code other than zero indicates an error	12
Message qualifier (output)	3 chars	The error message prefix	13
Message id	Full word	The message ID	14

Field	Size	Description	Parameter Number
(output)			
Reason (output)	80 chars varying	The short message text	15

Notes:

- The stored procedure is defined as General with NULLS.
- An indicator variable must be set for all output parameters.
- Set all Output variable indicators to -1.

  Setting output variable indicators improves performance, as these fields are not transmitted to the stored procedure.

DBXSEC Example

```
Security validation:

/* ASSIGN ALL THE PARAMETERS REQUIRED FOR THE SECURITY        */ /* VALIDATION STORED
PROCEDURE SUCH AS THE EXAMPLE BELOW THAT    */
/* SHOWS ASSIGNMENT TO FOUR OF THE PARAMETERS.                    */

COMMAND          = 'SELECT';
FUNCTION         = 'EDIT';
ENT_TYPE         = 126;
TSO_LOGON        = 'USER1';

/* SET UP INDICATORS */

WINDOWIND    = 0;
FUNCTIONIND  = 0;
SUBFUNCIND   = 0;
COMMANDIND   = 0;
ENTTYPEIND   = 0;
STATUSIND    = 0;
TSOIDIND     = 0;
DIALOGIND    = 0;
EXPROCIND    = -1;
EXTYPEIND    = -1;
EXEXECIND    = -1;
RTCODEIND    = -1;
MSGQUALIND   = -1;
MSGIDIND     = -1;
REASONIND    = -1;
```

```
/* CALL SECURITY VALIDATION STORED PROCEDURE */

EXEC SQL
  CALL DBXSEC(
     :CMD_INFO.WINDOW           INDICATOR  :WINDOWIND,
     :CMD_INFO.FUNCTION         INDICATOR  :FUNCTIONIND,
     :CMD_INFO.SUB_FUNC         INDICATOR  :SUBFUNCIND,
     :CMD_INFO.COMMAND          INDICATOR  :COMMANDIND,
     :CMD_INFO.ENT_TYPE         INDICATOR  :ENTTYPEIND,
     :CMD_INFO.STATUS           INDICATOR  :STATUSIND,
     :CMD_INFO.TSO_LOGON        INDICATOR  :TSOIDIND,
     :CMD_INFO.DIALOG           INDICATOR  :DIALOGIND,
     :CMD_INFO.EXIT_PROC        INDICATOR  :EXPROCIND,
     :CMD_INFO.EXIT_TYPE        INDICATOR  :EXTYPEIND,
     :CMD_INFO.EXIT_EXECUTION   INDICATOR  :EXEXECIND,
     :RET_CODE                  INDICATOR  :RTCODEIND,
     :MSG_QUAL                  INDICATOR  :MSGQUALIND,
     :MSG_ID                    INDICATOR  :MSGIDIND,
     :REASON                    INDICATOR  :REASONIND);

IF SQLCODE ^= 0
  THEN RETURN_CODE = SQLCODE;
  ELSE DO;
     IF RET_CODE ^= 0
       THEN RETURN_CODE = RET_CODE;
       ELSE RETURN_CODE = 0;
END;
```

Get_streamtext_32K

Get_streamtext_32K is a DB2 function that displays text for a repository instance as if it was a single line of text up to a maximum of 32000 characters.

How it works:

1. The Entity ID, Entity type, and Text table number is passed to the function as character strings.
2. The function retrieves all of the text meeting the input criteria and strings it together into a single 32000-byte field returning this field to the user. All trailing spaces except for one is trimmed from each row returned from the text table.

Note: The result set of an SQL statement containing an order-by clause cannot exceed 32 KB. If you must use an order-by clause in your SQL and additional columns in the result set exceed 700 bytes, you can cast the output of the function to a smaller data type. For additional information, see the example section.

Input

The following table describes the input fields processed by the Get_streamtext_32K function:

Field	Size	Description	Parameter Number
Entity id	Varchar(13)	The entity id of the object being deleted	1
Entity type	Varchar(8)	The entity type of the object being deleted	2
Text Type	Varchar(2)	The text table number 1 through 5	3

Output

The following table describes the output fields produced by the Get_streamtext_32K function:

Field	Size	Description
Text	Varchar(32000)	The text streamed together to form a single field

Note: The input must be in character format. All Input fields are required.

GET_STREAMTEXT_32k Example

```
Function CALL
```

Get the name, status, version, and text from text table 1 for the instance that has an ent_id of 95638.

```
SET CURRENT SQLID = 'DBXREL30';

SELECT E.ENT_ID, E.NAME, E.STATUS, E.VERSION,

      DBXREL30.GET_STREAMTEXT_32K(CHAR(E.ENT_ID), CHAR(X.ENT_TYPE), '1')

      AS TEXT1

   FROM DBXREL30.DBX_DBX_IEW_ATTR_TP E, DBXREL30.DBX_XREF X

   WHERE E.ENT_ID = XENT_ID

     AND E.ENT_ID = 95638;
```

Text1 will be a column containing up to 32000 bytes of text.

To return less text you can cast the output of the function to a different data type.

For example, to return the text from all five tables when you know that no single instance contains more than 3000 bytes of text, you could issue the following SQL statement:

```
SET CURRENT SQLID = 'DBXREL30';


SELECT E.ENT_ID, E.NAME, E.STATUS, E.VERSION,

     VARCHAR(DBXREL30.GET_STREAMTEXT_32K(CHAR(E.ENT_ID),

             CHAR(X.ENT_TYPE), '1'), 3000) AS TEXT1,

     VARCHAR(DBXREL30.GET_STREAMTEXT_32K(CHAR(E.ENT_ID),

             CHAR(X.ENT_TYPE), '2'), 3000) AS TEXT2,

     VARCHAR(DBXREL30.GET_STREAMTEXT_32K(CHAR(E.ENT_ID),

             CHAR(X.ENT_TYPE), '3'), 3000) AS TEXT3,

     VARCHAR(DBXREL30.GET_STREAMTEXT_32K(CHAR(E.ENT_ID),

             CHAR(X.ENT_TYPE), '4'), 3000) AS TEXT4,

     VARCHAR(DBXREL30.GET_STREAMTEXT_32K(CHAR(E.ENT_ID),

             CHAR(X.ENT_TYPE), '5'), 3000) AS TEXT5
```

```
FROM DBXREL30.DBX_DBX_IEW_ATTR_TP E, DBXREL30.DBX_XREF X

WHERE E.ENT_ID = XENT_ID AND E.ENT_ID = 95638;
```

Chapter 14: The Batch Load Facility

One of the most important utilities available within the CA Repository tool set is the Batch Load Facility. The Batch Load Facility is used to interface with the CA Repository in a batch mode. This facility performs SELECT, INSERT, UPDATE, REUSE, and DELETE actions for entities, associations and relationships using either data files or SQL statements to provide the necessary input data.

Though most often used to insert new metadata into the repository, it can also be used to modify, retrieve, or delete existing metadata from the repository.

The Batch Load Facility is capable of both loading any metadata that can be captured, and extracting any metadata that can be identified.

This facility will not only allow you to design highly-specialized CA Repository import and export jobs, but also will give you a better understanding of some of the repository-designed functionality.

This section contains the following topics:

The Batch Load Statement (see page 227)
Batch Load Syntax Scenarios (see page 243)
Workstation Processing with Batch Load (see page 250)
Sample JCL (see page 251)
Text Load (see page 253)

The Batch Load Statement

The various parameters required by the Batch Load Facility are specified in a Batch Load card. This load card, which must follow the format discussed below, must be supplied for every meta-entity (entity, association, or relationship type) that is the subject of a Batch Load Facility job.

The layout for the CA Repository Batch Load syntax for INSERT, UPDATE, and DELETE is shown in the illustration that follows.

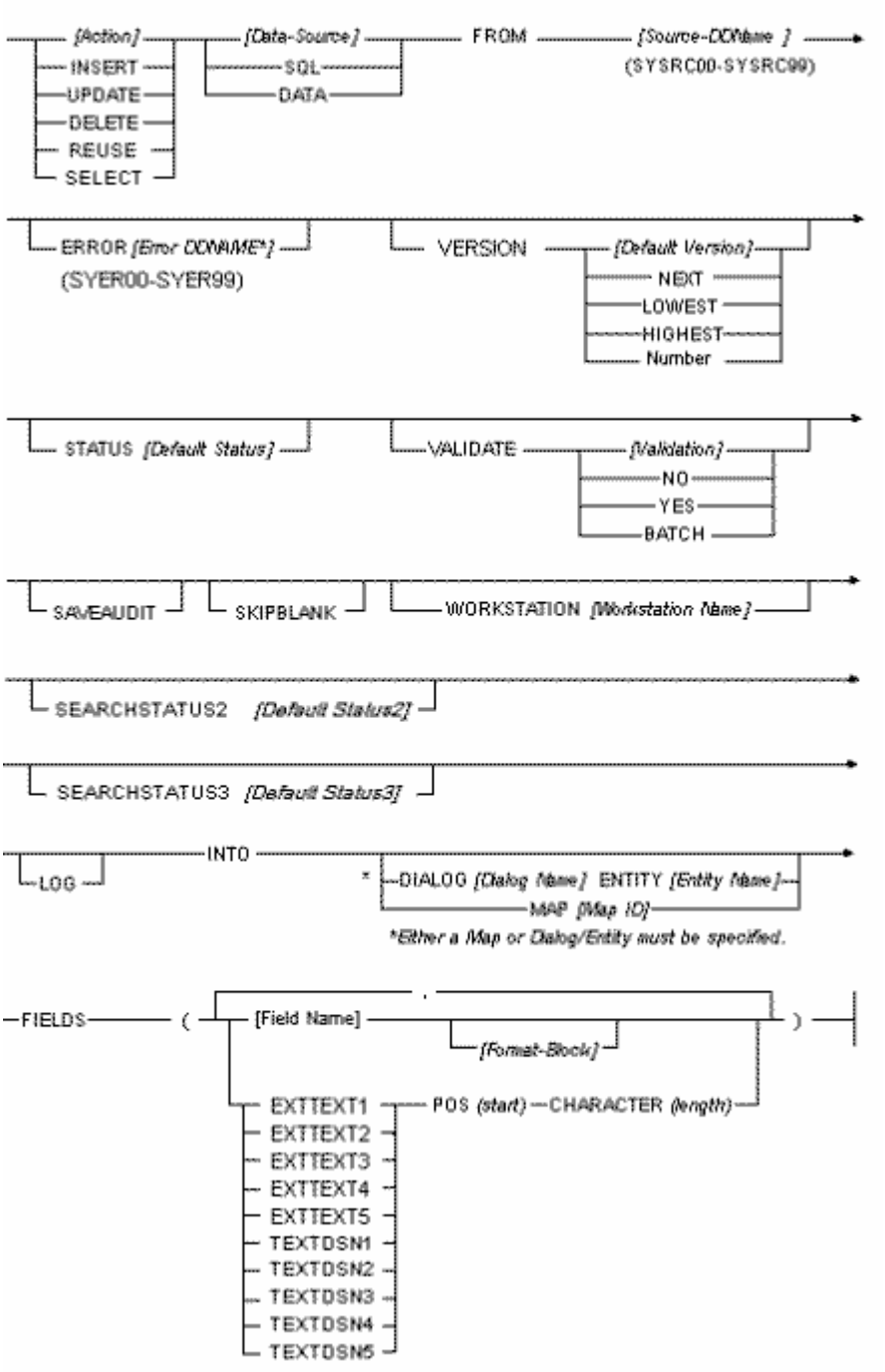
Keywords are in uppercase letters and should be entered exactly as shown (all valid abbreviations will be discussed later in this chapter). User-supplied variables are mixed case and enclosed by brackets.

Samples of batch load syntax are provided later in this chapter.

The following is an example of the layout of the Batch Load Syntax:

```
[Action] [Data-source] FROM [Source-DDName] ERROR [Error-DDName]
VERSION [Default-Version] STATUS [Default-Status] VALIDATE [Validation]
SAVEAUDITWORKSTATION [Wksn-Name] LOG
INTO DIALOG [Dialog-Name] ENTITY [Entity-Name]
FIELDS ([Field Name] [Format Block];
        [Filed-Name [Format Block];)
```

Repository LOAD Syntax for INSERT, UPDATE, DELETE, REUSE, and SELECT



Required Batch Load Syntax Parameters

This section describes the required parameters used in the batch load statement.

Action Parameter

The *Action* parameter must be specified. It is the action INSERT, UPDATE, DELETE, SELECT, or REUSE that is to be performed. These actions are fundamental to the batch load process.

Note: The REUSE parameter can be used in place of the INSERT and UPDATE parameters in all situations. See the following table.

Parameter	Definition
INSERT	Indicates that metadata will be loaded into the repository
UPDATE	Indicates that meta-data already stored in the repository will be changed
DELETE	Indicates that repository metadata will be deleted
SELECT	Indicates that repository metadata will be exported Be aware that the repository metadata will remain in the repository; it is not removed or changed.
REUSE	Combines INSERT and UPDATE logic in a single command New objects will be inserted and existing objects will be updated.

Data Source Parameter

The Data-Source parameter must be specified; it indicates what type of data is being used as input.

There are two acceptable types of input for the data-source parameter:

- DATA

  If DATA is specified, the batch load facility inputs a flat file made up of rows (records). This data set is made up of one or more records that will be processed individually.

- SQL

  If SQL is specified, the input file will contain an SQL SELECT statement. The batch load facility executes the SQL SELECT statement producing a temporary output file (TEMPSQL) that is then used as input-as if DATA had been specified. When using SQL as the input source, there are several considerations:

  – It is important for the TEMPSQL data set to be allocated correctly. CA suggests using the repository-supplied sample batch load syntax, in the BTCHLOAD member of the CCP$JCL library, as an example for your batch load jobs.

  – If the SQLTEMP data set proves too small for the batch load output, then simply increase the allocation.

  – The user may want to inspect this TEMPSQL data set to investigate the exact positioning of the metadata that results from the SQL. The positioning of SQL results must coordinate precisely with the Field Declarations of the Batch Load syntax.

  When using an SQL statement for input, the resultant data provided by the SQL statement will be the concatenation of all returned data fields. No conversions are performed in this process, so appropriate care must be exercised when writing the SELECT statement to ensure that the data is returned from DB2 in the correct format. For example, all literals in the SELECT statement will be returned as varying character strings: if a varying string is not desired, you will need to use the DB2 function SUBSTR in the SELECT statement to force DB2 to return a fixed length string.

  The DB2 functions INTEGER and DECIMAL are also very useful to put the selected data into the correct format (for further information, see the IBM *SQL Reference Guide*).

FROM (Source-DDName)

The keyword/literal FROM *is* a required component of the Load statement. It is used to indicate that the data immediately following is the Input DD Name, or the Source-DDName parameter:

- The Source-DDName parameter specifies the file that contains either the data to be processed or the SQL statement that is to run.
- The LRECL of a Source-DDName containing an SQL statement must be 80 bytes. For input data files, the LRECL must be less than or equal to 4044 bytes.
- Valid values for the Source-DDName are SYSRC00 through SYSRC99. The actual data set name is specified by a DD statement in the JCL. Optionally, the DD can be an instream data set.

Optional Batch Load Syntax Parameters

The next set of parameters are all optional. These parameters control the batch load.

ERROR [Error-DDName]

The keyword Literal ERROR indicates that immediately following is the Error-DDName.

Notes:

- The Error-DDName refers to the file that will be used as an error file. If for any reason, records are rejected from the load, they will be written to the error file.

  Valid values for the Error-DDName are SYSER00 through SYSER99.

- The most common reason for Batch Load rejections are "not found" errors. This results when the criteria specified by SQL or field declarations return no rows. Other possible errors include -811s (two or more rows meet the given criteria), insufficient authority for the Batch Load action, and validation errors (code table, policies, and so on.).

- The error data set should be specified in the JCL.

- The error data set's LRECL must be equal to the LRECL of the input data set (Source-DDName).

  If used with SQL input, its LRECL must be as wide as necessary for the results of a SELECT statement. If an error occurs, the Batch Load Facility writes the offending record to the error file and a message to the SYSPRINT. The message gives an indication of why the record was rejected, the record number (sequence in the Batch Load processing) and the total number of records rejected so far in the processing.

- Because the error file contains all records rejected during the load process, including records rejected when an SQL statement is specified as the *Data-Source*, the error file can be edited and then resubmitted as the source data file in a new Batch Load job. This process can be repeated until all records are processed. Keep in mind that if the data source is SQL, each error file record has the format of the concatenation of all fields selected by the SELECT statement.

VERSION [Default-Version]

A default version number which will be used for all blank versions found in the data file, including those of the source and target entities, can be specified after the keyword VERSION.

Notes:

- Default-Version can be specified as a valid version, 0 through 32767.
- Additionally, the parameters LOWEST, HIGHEST, or NEXT can be specified as the Default-Version. These parameters cause the Batch Load Facility to search for the lowest-available, highest-available, or next-available version when inserting records with no version specified in the field declaration or blank versions in the data file. If you are processing Batch INSERT, these parameters ensure that a record is inserted even if another occurrence exists with the same name and status attributes.
- Using the Default-Version parameter also works with Batch UPDATE and DELETE for records processed with blank version fields.
- When you are performing Batch INSERT, specify a Default-Version of 0 if you want duplicate records to be rejected.
- Specify NEXT (or LOWEST or HIGHEST) if you want Name, Status duplicates to be inserted.
- Batch INSERT considerations:
  - If no VERSION [Default-Version] parameter is specified, then records will be inserted exactly as indicated by the FIELD parameter for version attributes.
  - If no FIELD parameters are specified for version attributes **and** no VERSION [Default-Version] parameter is specified, processing terminates with an error.

STATUS [Default-Status]

The default status can be specified following the keyword STATUS and is used as the primary status. New rows are inserted into the primary status. Updates are performed in the primary status if the row is found.

Notes:

- Default-Status should be a valid CA Repository status.
- If no STATUS [Default-Status] parameter is specified, records are processed exactly as indicated by the FIELD parameter for status attributes.
- If no FIELD parameters are specified for status attributes **and** no STATUS [Default-Status] parameter is specified, then processing continues based on the disposition of the VALIDATE parameter. VALIDATE BATCH or VALIDATE NO insert a space while VALIDATE YES returns an error message indicating that status is a required attribute.
- Any status containing spaces must be placed in single quotes.

VALIDATE [Validation]

As part of the batch load process, all records can be validated to ensure compliance with existing CA Repository code tables, status, or policy restrictions.

Note:

- The valid values for the Validation parameter are YES, NO, and BATCH. In the batch load syntax, the keyword VALIDATE should be followed by one of these three parameters. If no Validation parameter is specified, then VALIDATE defaults to YES.
  - If YES is specified for Validation, the meta-data is processed by the Batch Load Facility to verify that repository occurrences being processed meet all of the standards currently designated in the CA Repository (for example, code tables validation, policies, and so on.). When YES is specified, expect a significant negative impact on the performance of the job.
  - If NO is specified for Validation, the Batch Load Facility makes no verification of the aforementioned standards.
  - BATCH is the equivalent of NO.
- In order to set VALIDATE to BATCH or NO for the Batch Load Facility (as with all CA Repository batch jobs), the user must belong to a CA Repository privilege group that has access to the following command:

Window	Function	Command
BLOAD	BLOAD	NOVALID

Note: As explained earlier, the Validation options NO and BATCH prevent repository standards checking during Batch Load Facility processing. These are the most common settings for Validation, and with good reason. Repository administrators often want all occurrences to be inserted regardless of whether they meet repository specifications. Repository standards are much more easily enforced on a repository occurrence than on an object stored outside of the CA Repository.

SAVEAUDIT

The SAVEAUDIT parameter specifies that all audit attributes (those with a map ATTR_ CODE of C) will **not** be set to the current time, date, and/or User-ID, but will be set to the values from the SQL statement or data file.

This parameter is primarily for use by the repository CASE Tool interfaces, but it can also be used during initial conversion of data from other sources.

SKIPBLANK

The SKIPBLANK parameter is used to support loading entity types with on-screen ties to other entity types. When set, any relationship or association that has a blank source or target name will be *skipped* during processing (that is, not processed).

This arrangement allows an input file containing both the entity information and tie information to be processed correctly with two sets of load syntax: The first set loads only the entity information. The second set would subsequently load the associations and relationships, skipping all those records that did not specify a tie to be created.

SEARCHSTATUS2

SEARCHSTATUS2 specifies a second status to search if the object is not found in the primary status.

Notes:

- The status specified as SEARCHSTATUS2 should be a valid CA Repository status.
- SEARCHSTATUS2 is only valid when the action is REUSE.
- Any status containing spaces must be placed in single quotes.

SEARCHSTATUS3

SEARCHSTATUS3 specifies a third status to search if the object is not found in the primary status or secondary search status.

Notes:

- The status specified as SEARCHSTATUS3 should be a valid CA Repository status
- SEARCHSTATUS3 is only valid when the action is REUSE
- Any status containing spaces must be placed in single quotes

WORKSTATION [Wksn-Name]

The WORKSTATION parameter is used to specify the workstation into which occurrences are added during batch load execution.

Be aware of the following:

- The Wksn-Name must exist within the repository to use the WORKSTATION parameter
- If the Wksn-Name contains spaces, it should be enclosed by single quotation marks

The WORKSTATION option is recommended and always used during CA Repository CASE Interface processing.

LOG

The keyword LOG can be used to force I/O logging.

Notes:

- If this parameter is not included, logging will default to the value specified on the map.
- If LOG is specified, expect a significant negative impact upon the performance of the job.

  Consequently, CA does not recommend activating the LOG mechanism.

  The LOG parameter activates a repository logging mechanism to record the load affect. The LOG parameter does **not** allow batch load results to be *rolled back*. If you are performing an INSERT action, then loading into a workstation will always allow you to roll back a batch load process. For a discussion of Workstations, see either the "Workstation" chapter of the *CA Repository for z/OS Product Guide* or the Workstation Processing with Batch Load section of this chapter.

Destination Parameters

The next set of parameters required for batch is INSERT, UPDATE, DELETE, and REUSE. The destination refers to the location in the CA Repository model that will be processed with INSERT, UPDATE, DELETE, or REUSE statements. For a discussion of Batch SELECT syntax, see the Batch SELECT section later in this chapter.

INTO

The keyword INTO indicates that the *destination* parameters are to follow. At this point, you have two choices; you can specify an entity type and dialog or a map.

ENTITY [Entity-Name]

The ENTITY parameter indicates which entity type is processed. The Entity-Name should specify the name of an existing repository entity type.

The DIALOG and ENTITY parameters are unnecessary if the MAP parameter is used.

Note: If the entity type or dialog name has any embedded spaces, you need to enclose the name in single quotes (for example: PHY AL).

DIALOG [Dialog-Name]

When using the ENTITY parameter, the DIALOG parameter must also be used. This Dialog-Name parameter is used to indicate the specific dialog in which the processing will occur and should be a valid CA Repository dialog.

If DIALOG is to be specified, it must be specified *before* the ENTITY parameter.

MAP [Map-ID]

The MAP keyword literal indicates that the Map-ID of the entity type being processed will follow.

The Map-ID should represent the identifier of an existing repository map. If this parameter is used, the specified map will be used to perform the load.

Note: If the MAP parameter is used, you do not need the DIALOG and ENTITY parameters.

Field Declarations

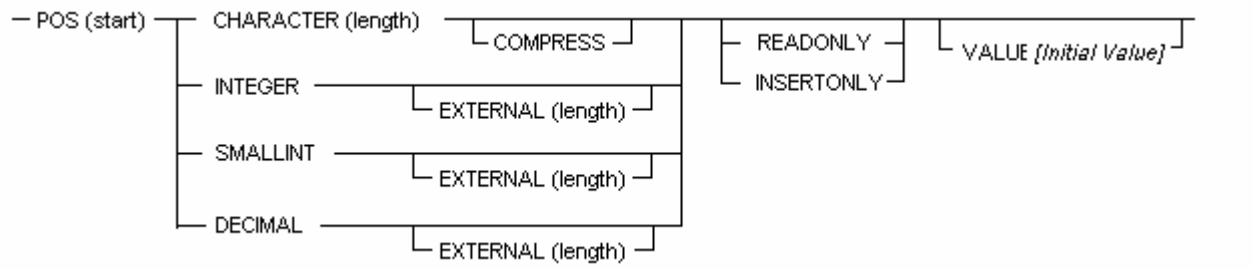
The last group of parameters indicates the specific input fields of the selected entity type into which the data is to be loaded.

FIELDS ([Field-Name] [Format-Block])

The keyword FIELDS is used to indicate the beginning of the field declarations. Ordering of field declarations in the FIELD section has no effect on the results of the batch load job.

Note: The Field-Name parameter refers to the DB2 column name of the attribute type to be processed. The name attribute of the entity type to be processed is the only "required" attribute for batch load processing.

- With the connecting entity types (Associations and Relationships), if the attribute in the FIELDS parameter does not belong to the entity type but to the source or target of the entity type, the attribute must be prefixed with S. or T.
- The Format-block section of the syntax (see the following figure) is for specifying the format of the data as it exists in the metadata input file. The POS [Start] parameter is used to specify the position (ISPF column number) in which the data pertaining to a particular field begins, relative to one. Care should be taken in setting Start parameters for fields processed through the Data-Source SQL.



Abbreviations for the Format Block are listed below.

Keyword	Abbreviation
CHARACTER	CHAR
INTEGER	INT
SMALLINT	SMALL
DECIMAL	DEC
EXTERNAL	EXT
READONLY	READ
INSERTONLY	INSERTSONLY
VALUE	VALUE

- [Data-Type] ([Length]) parameter

  This parameter is used to specify the data type of the input data field being processed and the number of characters being used as input to the Batch Load syntax.

  Valid Data-Types are CHARACTER, SMALLINT, INTEGER, and DECIMAL. The Batch Load Facility expects the input data field to be in the Data-Type specified. Portions of the data file record that are not referenced by a field declaration of the Batch Load syntax will be ignored. Also known as "Data File Gap", they do not interfere with Batch Load Facility processing.

- EXTERNAL

  The EXTERNAL flag can be used when Data-Type is specified as INTEGER, SMALLINT, and DECIMAL.

  EXTERNAL will indicate that a numeric value is stored in a character format in the data source file. Upon INSERT or UPDATE, the field value will be converted from character format to numeric format. If the EXTERNAL indicator is not specified for SMALLINT, INTEGER, or DECIMAL fields, then the field values are expected in Hexadecimal format.

  The Batch Load Facility will, however, perform data type conversions without the use of the EXTERNAL indicator. The facility will determine the data type of the DB2 column being processed and make the appropriate conversion from the character data type. For example, if loading from an integer stored in character format into a column with a data type of integer, CHARACTER (5) will process in the same manner as INTEGER (5) EXTERNAL.

  For ease of use, CA recommends that you specify CHARACTER as the data type for field declarations and allow the Batch Load Facility to make the appropriate data type conversion.

  Note: Unless Date, Time, and Timestamp are system-generated, they must be included in the input.

- The keyword literal COMPRESS indicates that all spaces, " ", for a field declaration should be removed during processing.

  COMPRESS is only available for character fields. For example, if you were performing a Batch INSERT of an occurrence with a name field starting in position one with a length of 40 characters, such as:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
CUSTOMER     .      CUST-ID
```

  Then the occurrence name would be processed as:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
CUSTOMER.CUST-ID
```

- READONLY

  When performing a batch UPDATE or REUSE, some fields should be followed by the keyword READONLY to indicate that they are to be used as the unique identifier for the records being processed.

  The READONLY fields will not be updated with Batch UPDATE or REUSE.

  If a key field needs to be updated, that field should be referenced twice, as follows:

  – In the key field with a READONLY indicator

  – In the modified field with no READONLY indicator

  When processing DELETE, UPDATE, and REUSE actions, some fields *must* be specified as READONLY to correctly determine which entity occurrences should be processed. If no fields are specified as READONLY, an error will occur during batch load processing. See the sections on Batch UPDATE, Batch DELETE and Batch REUSE for examples of the READONLY parameter.

- INSERTONLY

  When performing an UPDATE or INSERT with the REUSE keyword, some fields should be followed by the keyword INSERTONLY to indicate that they are to be used only if the row is being inserted.

  See the sections on Batch REUSE for examples of the INSERTONLY parameter.

- VALUE

  When performing an INSERT with the REUSE keyword, the attribute with the VALUE clause will be populated with this initial value.

  Note: VALUE is only valid for those fields defined as INSERTONLY when using the REUSE keyword.

Special Keywords (TEXTDSNn) in Field-Name

Special keywords (that is, TEXTDSN1, TEXTDSN2, etc.) placed in the Field-Name parameter are used to denote that extended text should be processed.

If specified, the load will use the given position and length to reference the data set (sequential or partitioned) where the extended text resides. Therefore, each entity occurrence that is to have extended text must reference a data set containing extended text. Data set names should be fully qualified with quotes, and should include the member name if applicable. Extended text can be specified for all five text sets. Text processing occurs after the action on the entity occurrence is performed.

Note: CA recommends use of the Extended Text Load facility, referenced in the Text Load section, instead of Batch Load Facility's extended text load functionality.

System Generated Attributes

It is possible that an attribute has some special processing or format rule specified in the CA Repository map such as attribute concatenation. Attribute concatenation processing will be performed if an attribute is mentioned in the field declarations, and that attribute's field declaration points to a space (that is, a blank location within the metadata source). If the attribute is not mentioned or the POS parameter points to non-blank metadata within the source data file, then attribute concatenation processing will not be performed. Instead, the metadata from the source data file will be processed.

All attributes that make up the concatenation must be included in the field statement. For example in the DB2 Dialog, column name is generated from the concatenation of the source table name and the target DB2 column name. S.TB_NAME must point to non-blank metadata in the source data file. T.DB2_COLUMN_NAME must point to non-blank metadata in the source data file. COLUMN_NAME can now point to a blank position in the source data file.

Special Keywords (EXTTEXTn) in Field-Name

Special keywords (that is, EXTTEXT1, EXTTEXT2, etc.) placed in the *Field*-Name parameter are used to denote that extended text should be processed.

If specified, the load will use the given position and length to reference the text data in the input file. Extended text can be specified for all five text sets. Text processing occurs after the action on the entity occurrence is performed. EXTTEXT keywords are only valid when using the REUSE option.

Sample Batch Load Syntax

Shown below is an example of batch INSERT syntax for the ELEMENT Entity Type. Notice the use of EXTERNAL as a format block parameter.

```
INSERT DATA FROM SYSRC00 ERROR SYSER00
VERSION 99 STATUS DBXS VALIDATE BATCH
INTO DIALOG DB2 ENTITY ELEMENT
FIELDS
  (ELEMENT_NAME  POS(1)  CHAR(10),
   STATUS        POS(11)  CHAR(8),
   VERSION       POS(19)  CHAR(6),
   DATA_TYPE     POS(26)  CHAR(5),
   LENGTH        POS(31)  INTEGER EXTERNAL(3));
```

This Batch INSERT Syntax would be valid with the following SYSRC00.

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
CUST-ID   DBXP    0     NUM  009
PROD-CD                  CHAR 004
STATE-CD  DBXT    0     CHAR 002
```

In this example, the Default-Status, DBXS, and Default-Version, 99, would be used only for the PROD-CD element. As mentioned earlier, the parameters STATUS and VERSION are processed only when the batch load syntax for the fields Status and Version point to spaces or when the Status and Version field declarations are absent.

Notice also that the Length field is converted by the Batch Load Facility from a three-byte character value to an integer value by the EXTERNAL parameter. Because the Length column is defined as an integer within the repository, this conversion is required.

Additional Notes

The SYSPRINT file contains all batch load processing messages from the load. All rejected records generate an error message, the input file record number, and the error file record number. A summary of the number of records read and rejected appears at the end of the SYSPRINT file. The return code of the load step will be *0* if no errors occur, *8* if only rejections occur, and greater than four if processing is stopped for any other reason (SQL error, file error, etc.).

Default values for attributes and system generated attributes are not processed by Batch Load syntax unless the attribute is included in the Field list and is pointed to an empty space in the input file. System Generated attributes and default values for attributes are processed when occurrences are accessed on-line. System Generated attributes are not validated unless Validation is set to YES. As mentioned previously, concatenated attributes are processed when loaded with a space, " ". For additional information on maps, see the Extending with the Extend Model chapter of this manual.

SQL -811

One common error returned when using batch UPDATE or batch DELETE is the SQL -811. An SQL -811 indicates that during processing "more than one row was returned, when only one was expected." If you get this error, then it means that more than one occurrence meets the current criteria for the UPDATE or DELETE statement. For example, if the input file for a batch update indicated both the name and status attributes, then the UPDATE statements would proceed without error until it attempted to process two occurrences with the same name and status, that is, different versions of the same occurrence. To guarantee that occurrences are uniquely identified by Batch Update or Delete, use all three key attributes, Name, Status and Version, in the FIELD parameters.

In order to assist in any possible troubleshooting, you may be asked to run Batch Load with *Debug*." To turn on the CA Repository Debug statements, find the following line in the JCL: PARM = ("/N"). Change the N to "/Y." The debug statements will be written to the data set specified in the SYSPRINT file.

Reference a Single SYSIN with Multiple Load Statements

Multiple load cards can reference a single SYSIN when the Data-Source is specified as SQL. The Batch Load Facility processes the LOAD statements and SQL statements sequentially, searching for semicolons (;) to indicate the ends of SQL statements and Batch Load Statements. The CA Repository ADW/IEW Interface uses single SYSIN Batch Load processing for repository population.

Batch Load Syntax Scenarios

This section discusses the application of the following batch load syntax options: INSERT, UPDATE, DELETE, and SELECT. Additionally, it presents the nuances of batch load syntax when applied to the different entity types: Entities, Relationships and Associations. If you have additional questions, contact CA Technical Support.

Batch INSERT: Load Metadata

The most common use of the Batch Load Facility is to load metadata into the CA Repository. If this is the intended use of the Batch Load Facility, then the *Action* parameter must be set to INSERT. The metadata source type (DATA or SQL) is specified in the Data-Source parameter followed by the FROM literal. The DD of the input data set is specified as the Source-DDName parameter. Optionally, the ERROR parameter may be specified, accompanied by the Error-DDName. Any records rejected during the INSERT will be sent to the data set specified in the Error-DDName parameter. A rejection may occur if an occurrence already exists and the Default-Version parameter is not set to LOWEST, HIGHEST, or NEXT. An Association or Relationship occurrence may also be rejected if the source or target occurrence is not found in the repository.

The Destination of the metadata to be loaded should be specified by the DIALOG and ENTITY parameters, or the MAP parameter. Next, the Batch INSERT Syntax requires FIELD parameters to specify the repository columns that will be populated.

Listed below is an example of Batch INSERT Syntax for the TABLE entity type:

```
INSERT DATA FROM SYSRC00 ERROR SYSER00
VERSION NEXT STATUS DBXT VALIDATE BATCH
INTO DIALOG DB2 ENTITY TABLE
FIELDS
  (TB_NAME             POS(1)    CHAR(10),
   STATUS              POS(11)   CHAR(8),
   VERSION             POS(19)   CHAR(6),
   CREATOR             POS(25)   CHAR(8),
   LONG_DESCRIPTION    POS(33)   CHAR(51));
```

Note: Omitting these fields will cause a -180 SQL ABEND.

This Batch INSERT Syntax would be valid with the following SYSRC00.

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
CUSTOMER  DBXP    2   PRMVS   THIS TABLE HOLDS DATA ABOUT CUSTOMERS
PRODUCT   DBXT    1   TEST    THIS TABLE CONTAINS DATA ABOUT OUR PRODUCTS
EMPLOYEE  DBXT    0   PROD    THIS TABLE CONTAINS DATA ABOUT OUR EMPLOYEES
```

Load Metadata into Associations and Relationships

When loading metadata into Associations or Relationship types, either the key attributes or a set of attributes which uniquely identify the Source and Target occurrences being tied together must be supplied to the Batch Load Facility. If the metadata is destined for a Relationship type, then the key attributes of the relationship itself must accompany the source and target key attributes. The source and target status/version can be hard-coded using the Default-Status and Default-Version parameters if the field declaration points to a space or is not specified.

Listed below is one example of Batch Load Syntax for loading an association, and another for loading a relationship.

Association:

```
INSERT DATA FROM SYSRC00 ERROR SYSER00
VERSION NEXT STATUS DBXT VALIDATE NO
WORKSTATION 'ASSOCIATION LOAD'
INTO DIALOG DB2 ENTITY 'PHY AL'
FIELDS
(S.ELEMENT_NAME    POS(01)  CHAR(18),
 S.STATUS          POS(19)  CHAR(08),
 S.VERSION         POS(27)  CHAR(06),
 T.ELEMENT_NAME    POS(33)  CHAR(18),
 T.STATUS          POS(51)  CHAR(08),
 T.VERSION         POS(59)  CHAR(06));
```

Relationship:

```
INSERT DATA FROM SYSRC00 ERROR SYSER00
VERSION NEXT STATUS DBXT VALIDATE NO
WORKSTATION 'RELATIONSHIP LOAD'
INTO DIALOG DB2 ENTITY COLUMN
FIELDS
(S.TB_NAME         POS(01)  CHAR(10),
 S.STATUS          POS(11)  CHAR(08),
 S.VERSION         POS(19)  CHAR(06),
 T.ELEMENT_NAME    POS(25)  CHAR(10),
 T.STATUS          POS(35)  CHAR(08),
 T.VERSION         POS(43)  CHAR(06),
 T.DB2_COLUMN_NAME POS(25)  CHAR(10),
 COLUMN_NAME       POS(49)* CHAR(21),
 STATUS            POS(70)  CHAR(08),
 VERSION           POS(78)  CHAR(06));
```

Please note, the asterisk (*) indicates an attribute that could be pointing to a blank space in the metadata source, if that attribute is defined on the CA Repository map as a concatenated attribute with a default value (as many relationship names are).

Note: The asterisk (*) is for illustrative purposes only and should not be put into the batch statement.

Batch UPDATE: Change Metadata

The Batch Load Facility can also be used to update metadata that has already been inserted into the repository. If this is the intended use of the Batch Load Facility, then the Action parameter must be set to UPDATE. As before, the input for the UPDATE is specified by the Source-DDName parameter. Any records rejected during the UPDATE will be sent to the data set specified in the Error-DDName parameter.

If the intended action is to update meta-data, the key attributes (Name, Status, and Version) or any set of attributes that uniquely identify each occurrence being updated, should be identified as such within the FIELDS section of the Batch SQL UPDATE statement so that these attributes are used as keys instead of attributes to be updated. To accomplish this, add the keyword READONLY to the field declaration of each attribute being used as a key. The only other attributes that should be mentioned in the FIELDS parameter should be the attributes that will be updated.

A Batch UPDATE may be implemented with the following SYSIN and SYSRC00:

```
UPDATE DATA FROM SYSRC00 ERROR SYSER00
VERSION 0 STATUS DBXT
WORKSTATION 'BATCH UPDATE'
INTO DIALOG DB2 ENTITY ELEMENT
FIELDS
(ELEMENT_NAME  POS(01)  CHAR(18) READONLY,
STATUS         POS(19)  CHAR(04) READONLY,
VERSION        POS(23)  CHAR(06) READONLY,
ELEMENT_NAME   POS(29)  CHAR(18),
DATA_TYPE      POS(47)  CHAR(04));

----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
CUSTOMER-NO       DBXP1    CUSTOMER-NUM      NUM
PRO-CD            DBXT0    PROD-CD           CHAR
EMPLOYE-NUM                EMP-NUM           NUM
```

Batch REUSE: Combine Inserts and Updates

The Batch Load Facility can also be used to combine INSERTs and UPDATEs into a single step. REUSE will update instances already in the repository and insert new rows depending on the field criteria. As before, the input for the REUSE is specified by the Source-DDName parameter. Any records rejected during the UPDATE will be sent to the data set specified in the Error-DDName parameter.

The REUSE function has the added advantage over the update function of search two additional statuses for a particular instance before it is inserted.

A batch REUSE may be implemented with the following SYSIN:

```
REUSE DATA FROM SYSRC00 ERROR SYSER00
              VERSION NEXT STATUS 'DB2T' VALIDATE BATCH
              SAVEAUDIT
              INTO DIALOG 'DB2' ENTITY 'ELEMENT'
              SEARCHSTATUS2 'DB2QA' SEARCHSTATUS3 'DB2P'
  FIELDS (
     ELEMENT_NAME               POS(0010) CHAR(020)  INSERTONLY
    ,DB2_COLUMN_NAME            POS(0010) CHAR(020)  READONLY
    ,DATA_TYPE                  POS(0031) CHAR(010) READONLY
    ,LENGTH                     POS(0041) CHAR(005) READONLY
    ,JUSTIFY                    POS(0046) CHAR(001)
    ,LONG_DESCRIPTION           POS(0047) CHAR(020)
    ,SIGNED                     POS(4044) CHAR(1) INSERTONLY VALUE 'N'
  );
```

The process searches for an element with the specified DB2_COLUMN_NAME, DATA_TYPE and LENGTH in status DB2T then DB2QA, then DB2P for a match.

If the instance is found, attributes JUSTIFY and LONG_DESCRIPTION are updated. If the instance is not found, an instance is inserted into status DB2T with the next available version number. All attributes are populated including ones marked as INSERTONLY and READONLY. The attribute SIGNED is populated with a Value of N.

```
REUSE DATA FROM SYSRC00 ERROR SYSER00
             VERSION NEXT STATUS 'DB2T' VALIDATE BATCH
             SAVEAUDIT
             INTO DIALOG 'DB2' ENTITY 'COLUMN'
             SEARCHSTATUS2 'DB2QA' SEARCHSTATUS3 'DB2P'
  FIELDS (
          S.TB_NAME            POS(0010) CHAR(020) READONLY
         ,S.CREATOR            POS(0052) CHAR(008) READONLY
         ,T.DB2_COLUMN_NAME    POS(0031) CHAR(009) READONLY
         ,T.DATA_TYPE          POS(0040) CHAR(006) READONLY
         ,T.LENGTH             POS(0046) CHAR(006) READONLY
         ,NULLS                POS(0060) CHAR(001)
         ,COLUMN_NAME          POS(4044) CHAR(001) INSERTONLY
   );
```

Notes:

- Enough attributes must be in the Source and Target objects to find one and only one row
- Will look for source and target object in up to three search statuses
- All Attributes must be in the map specified in the Syntax statement
- Non Read only and non Insert only attributes are updated

Batch SELECT: Export Metadata

The Batch Load Facility can also be used to export data that resides within the repository or on any other DB2 table and write that data to a flat file. Any data exported is still stored within the repository.

If you intend to use the Batch Load Facility to export data, then the Action parameter must be set to SELECT. The Data-Source parameter must be set to SQL. DATA is **not** a valid Data-Source option for Batch SELECT. The SQL SELECT statement that determines which attributes of which occurrences are exported from the CA Repository should be placed in the data set specified in the Source-DDName. Next specify the keyword literal INTO. This designates that the following data set will contain the output from the SQL SELECT statement. The DD name of the output data set should be specified in the Error-DDName parameter.

Listed below is an example of Batch SELECT Syntax. An additional example can be found in the BTCHSEL member of the repository CCP$JCL library.

```
SELECT SQL FROM SYSRC00 INTO SYSER00;
```

The following SQL query, if used as the input for the Source-DDName parameter, would return the name, status, and version attributes of rows stored on the DBX_DDL_ELEMENTS table that have a data type of CHAR and a status of DBXT.

```
SELECT E.ELEMENT_NAME, E.STATUS, E.VERSION
  FROM DBXDDL.DBX_DDL_ELEMENTS E
WHERE E.DATA_TYPE = 'CHAR'
  AND E.STATUS = 'DBXT';
```

Batch DELETE: Delet Metadata

The Batch Load Facility can also be used to delete metadata that resides within the CA Repository. If this is the intended use of the Batch Load Facility, then the Action parameter must be set to DELETE. The SQL SELECT statement or DATA file that determines which occurrences should be deleted from the CA Repository and placed in the data set specified in the Source-DDName parameter.

As with updates, the key attributes (Name, Status, and Version) or a combination of attributes that uniquely identify a set of occurrences must be specified as READONLY in the FIELDS parameter. No other attributes need to be referenced by the field declarations.

Use the following syntax as an example:

```
FIELDS
(Name-Field     POS ([Start])  [Data-Type] ([Length])  READONLY
,Status-Field   POS ([Start])  [Data-Type] ([Length])  READONLY
,Version-Field  POS ([Start])  [Data-Type] ([Length])  READONLY);
```

Note: Both the STATUS *[Default-Status]* and VERSION *[Default-Version]* parameters can be specified in the Batch Load card for Batch DELETE processing. In other words, you may specify a Default-Status and Default-Version instead of field declarations or to override blank Status and Version field declarations.

Workstation Processing with Batch Load

The Batch Load Facility was designed to take full advantage of the CA Repository workstation concept. Any information inserted or updated by the Batch Load Facility isl automatically stored within the workstation specified in the Wksn-Name parameter.

Note: Workstation refers to any object within the "ALL WORK" set (Entity type 25112). This includes the following objects: WORKSTN , WORKPACK, WORKPROJ, WORKSUBJ, CATEGORY, STEWARD, and DOCUMENT.

The SQL select statement below demonstrates how that information can be extracted. This SQL statement extracts information for a particular entity type; that is, metadata stored on a specific repository table. Note that the column, workstation, and table names in lowercase should be customized.

```
SELECT E.Name, E.Status, E.Version
FROM Creator.Table_Name E
   , DBX_WORKSTATION_D WD
   , DBX_WKSN_XREF WX
WHERE WX.WKSN_ID = WD.ENT_ID
  AND WD.NAME = 'Workstation_Name'
  AND WD.ENT_ID = E.ENT_ID;
```

Delete Within Workstations

Another powerful feature of the Batch Load Facility is its capability to delete from the repository all of the occurrences within a certain workstation. This differs from the other workstation delete commands (available on-line through the OPTIONS function) that remove an occurrence from a workstation, but not from the repository. Special care should be taken when using the WORKSTATION DELETE Syntax because of the major effects it can have on the repository.

Note: This special Batch Load DELETE Workstation syntax deletes occurrences that are found only within the specified workstation from the repository. If an occurrence is located within two or more workstations, that occurrence is not deleted from the repository unless the Workstation force action is in effect. The workstation force parameter is set on the Parm= string in the JCL.

The syntax for this DELETE workstation job is:

```
DELETE WORKSTATION [WkstnName];
```

CA Repository-Supplied Examples of the Batch Load Facility

Some examples of the Batch Load Facility that can be used as templates for an individual site's Load statements are supplied. These examples are located within the CA Repository CCP$JCL data set, under the member names: BTCHLOAD, BTCHSEL, BTCHDEL, and DELWKSN.

Sample JCL

```
//*********************************************************************
//*           S A M P L E   B A T C H   L O A D   J C L               *
//*********************************************************************
//* DBXLOAD PARM STRING:
//* 1) DEBUG: Y = DEBUG ON, N = DEBUG OFF.
//* 2) SUBSYSTEM: SUBSYSTEM TO CONNECT TO OTHER THAN ONE LISTED IN
//*               DBXPARM.
//* 3) ENT NAME POS: POSITION IN THE INPUT DATA FILE OF THE ENTITY
//*                  TYPE NAME; USED WHEN ONLY ONE DATA FILE EXISTS
//*                  FOR ALL ENTITY TYPES BEING LOADED.
//* 4) 3NF TEXT TABLE: NAME OF THE 3NF TEXT TABLE IF LOADING FROM
//*                    3NF TABLES.
//* 5) 3NF TEXT USER: USERID USED IN THE 3NF TEXT TABLES TO SEPARATE
//*                   DATA BY WORKUNIT.
//* 6) CASE: L = LOAD DATA IN MIXED CASE; OVERRIDES MAP DEFINITIONS.
//* 7) WKWN FORCE DEL: FORCE THE DELETION OF EVERY INSTANCE CONTAINED
//*                    IN A WORKSTATION DURING A BATCH WORKSTATION
//*                    DELETE; EVEN IF THAT INSTANCE IS CONTAINED IN
//*                    ANOTHER WORKSTATION; Y = FORCE DELETE, N =
//*                    DELETE ONLY WHAT IS NOT IN ANOTHER WORKSTATION
//*                    (THE WAY DBXLOAD HAS WORKED IN THE PAST).
//*********************************************************************
```

```
//DBXLOAD EXEC PGM=DBXLOAD,PARM=('/ ')
//STEPLIB  DD  DSN=PRMVS.LOADLIB,DISP=SHR
//DBXPARM  DD  DSN=PRMVS.ISPPLIB(DBXPARM),DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//TEMPSQL  DD DSN=&&SQLTEMP,DISP=(NEW,DELETE,DELETE),
//            DCB=(LRECL=4044,BLKSIZE=16176,RECFM=FB),
//            UNIT=SYSDA,SPACE=(TRK,(50,20),RLSE)
//SYSIN    DD  *
INSERT DATA FROM SYSRC00 ERROR SYSER00
       VERSION 0  STATUS DB2T VALIDATE YES
  INTO DIALOG DB2  ENTITY TABLE
  FIELDS (TB_NAME           POS(0001) CHAR(18),
          CREATOR           POS(0019) CHAR(08),
          EDIT_PROC         POS(0027) CHAR(08),
          VALID_PROC        POS(0035) CHAR(08),
          NUM_OF_ROWS       POS(0043) INTEGER,
          LONG_DESCRIPTION  POS(0047) CHAR(242),
          AUDIT_ACTIVITY    POS(0289) CHAR(01),
          QUAL_TABLE_NAME   POS(0298) CHAR(01));
/*
//*********************************************************************
//*                     DATA AND ERROR FILES                          *
//*********************************************************************
//SYSRC00  DD DSN=USER.SYSRC00,DISP=SHR
//SYSER00  DD DSN=USER.SYSER00,DISP=SHR
//
```

The values specified in the JCL parameters are as follows:

Parameter	Value
Debug	Y or N
Subsystem	A value in this parameter indicates that the DB2 subsystem name is other then the one specified in the DBXPARM file
Entity name position	The start position in the input file that contains the name of the object being processed. This field must be used when the input file contains more than one object type.
3NF Text table	The value used by CA Repository Exchange for Erwin prior to CA Erwin release 4.0
3NF Text User table	The value used by CA Repository Exchange for Erwin prior to CA Erwin Release 4.0
Case	L-The definition for mixed case on the map should be ignored

Parameter	Value
	and all data is to be loaded as mixed case
Workstation force delete	Y-The object will be deleted from the repository when performing a workstation delete even if the object resides in more then one workstation
Commit Mode	R-When running a workstation, delete report on what will be deleted The delete is performed and then a rollback is issued. If Commit Mode contains a C, the changes will be committed to the database.
Product Delete Report	Y- When running a workstation delete, produce a report on objects that have been deleted and those objects that have not been deleted due to inclusion in more then one workstation. N-No report will be produced

Text Load

The CA Repository Text Load Facility, separate from the Batch Load Facility, can be used to load 80-character fixed length records into existing repository entities.

Each set of text for an individual occurrence must be separated by header records with one of the following formats:

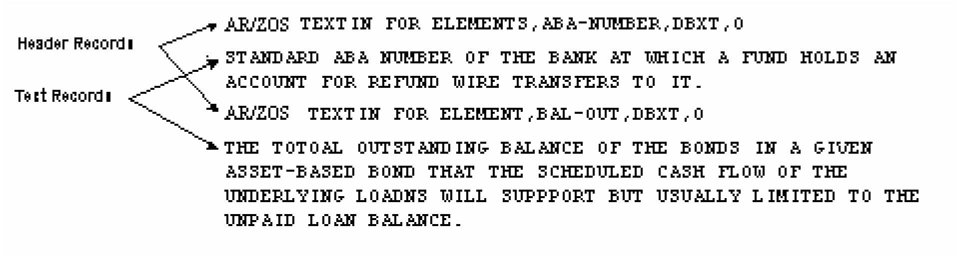
```
PR/MVS TEXTIN FOR [entity type,name,status,version]

DBEXCEL TEXTIN FOR [entity type,name,status,version]

AR/ZOS TEXTIN FOR [entity type,name,status,version]
```

The header record can be up to 500 bytes to support a key name of up to 254 characters.

This format contains the following components, each separated by a comma:

- The literal "AR/ZOS TEXTIN FOR"
- The entity type of the entity into which the text is to be loaded
- The name of the entity into which the text is to be loaded (8-character name)
- The status of the entity
- The version of the entity

All lines following the header record are considered text until another header record is encountered. Approximately 32,767 lines of text can be loaded for each text record, for each specified occurrence. The sample input file below depicts two contiguous header/text records.



Text Load Facility JCL

The JCL used to load the text records can be generated from the ARZUTIL menu.

To execute the command, enter TSO ARZUTIL from the command line.

```
---------------------------- LOAD TEXT ------------------------------
… COMMAND ===>                                                       …
…                                                     More:     +    …
…                                                                    …
… Enter Text load parameters                                         …
…                                                                    …
… ===> Debug Mode  : N         Y = Debug, N=No Debug                 …
… ===> Append  text: N         Y = Text will be appended             …
…                              N = Text will be replaced             …
… ===> Dialog Name : SYSTEM    Dialog name                            …
… ===> Text Table  : 1         1 thru 5 for the text table number    …
… ===> Status      : DBXT      Default Status                        …
… ===> Object      :           Object name where text will be loaded …
… ===> Upper Case  : N         Convert to Upper Case - Y or N        …
…                                                                    …
… ===> Create DB2 Load Input: N Y - DB2 Load input will be created   …
…                               N - Text loaded into text tables     …
… ===> Input Text dataset:                                           …
…           PRM.TXT.INPUT                                            …
… ===> DB2 Load Syntax dataset:                                      …
…                                                                    …
…                              The Syntax dataset must be an 80      …
…                              character dataset                     …
… ===> DB2 Load Data dataset:                                        …
…                                                                    …
…                              The Data   dataset must be an 105     …
…                              character dataset                     …
…                                                                    …
… ===> Disposition:            Disposition of DB2 load datasets      …
…                              NEW or SHR                            …
…                                                                    …
… Enter ENTER command to continue or END command to EXIT.            …
Ä¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢¢Ô
```

Values in the Load Text fields are as follows:

Field	Enter
Debug	Y or N
Append text	Y or N to indicate whether the loaded text overwrites, or is appended to, any existing extended text
Dialog Name	The name of the CA Repository dialog containing the entity into which the text is to be loaded
Text Table	The five extended text types to which the loaded text will be added
Status	A (default) status to be used for processing which will override all statuses specified in the header records of the input file This parameter can be left blank. If this parameter is left blank, the Extended Text Load Facility will use the status specified in each header record.
Object	An entity type to be used for processing which will override all entity types specified in the header records of the input file This parameter can be left blank. If this parameter is left blank, then the Extended Text Load Facility will use the entity type specified in each header record.
Upper Case	Y to convert name and text to uppercase, N to use mixed case
Create DB2 Load Input	Y indicates that instead of loading text directly into the text tables, DB2 load syntax and data will be created N indicates that the text will be loaded directly into the text tables
Input Text data set	The name of the input file containing the text data to be loaded.
DB2 Load Syntax data set	An 80-character data set that will contain the DB2 Load syntax
DB2 Load Data data set	A 105-character dataset that will contain the DB2 load data
Disposition	SHR-Both the DB2 load and Data data sets already exist and will be reused New-Neither the DB2 load nor Data data sets exist and both will be created

Chapter 15: Backup, Recovery, and Performance Tuning

This chapter explains performance, recovery, and security maintenance issues.

This section contains the following topics:

Control Tables and DB2 Tables (see page 257)
Backup and Recovery Procedures (see page 257)

Control Tables and DB2 Tables

The CA Repository for z/OS environment contains two distinct sets of DB2 tables:

- Control tables that store information about repository products
- Repository tables that store and retrieve entity attributes

Backup and Recovery Procedures

Backup and recovery procedures must be defined to ensure smooth and continued operation of the repository.

What Components Must Be Backed Up?

The following components must be backed up on a regular basis:

- CA Repository for z/OS DB2 Tables
- CA Repository for z/OS DB2 History Table
- CA Repository for z/OS Repository data sets

Back up ALL of your repository tables regularly and at the same time using a DB2 tool such as CA Unicenter DB2 utilities.

If you back up and restore only some of the tables, it will leave them "out of sync" as there is a row in the DBX_XREF table for every row in all other repository tables.

Note: As a general rule-if a table contains the column ENT_ID, it must be backed up when the DBX_XREF table is backed up.

Tune the Database

The repository is a DB2 application and therefore tuning techniques of any DB2 application apply to the repository.

Chapter 16: History Tables

Important! You must be using DB2 version 8 New Function Mode or later to use the features in this chapter.

History tables (audit trail) enable a user to track all updates and deletions for specific repository objects. The repository administrator uses the repository editor to specify which objects should have a History table built. History is kept in tables that are copies of the repository tables. Triggers are created for the objects. These triggers insert the *before* image of the event whenever an object is *updated* or *deleted*.

These tables are identical to the to repository tables with one exception-an additional column, ZTRIGEVENT, exists on the table. The value indicated in this column indicates whether a row in the corresponding repository table was updated (U) or deleted (D).

This section contains the following topics:

Enable the History Feature (see page 259)
Metadata History Object Selection (see page 263)

Enable the History Feature

Enabling the History feature consists of two distinct phases:

- Phase One: History file selection must be activated at the product level and the Product Level History process enables the framework for definition of history file selection at the metadata level.

  Activation at the product level generates the history audit tables for the core tables along with the associated triggers. For further details, see the Product Level History File Activation section of this chapter.

- Phase Two: Metadata History Object Selection process involves using the history feature on *selected, individual* metadata objects.

  During this process, you define the specific metadata objects that you want to audit. It is only when the history selection is activated at the metadata level that the corresponding history table for the specific object is created and its associated trigger. For details, see the Metadata History Object Selection section later in this chapter.

History Core Tables

The history core tables are defined as:

- DBX_TEXT_1
- DBX_TEXT_2
- DBX_TEXT_3
- DBX_TEXT_4
- DBX_TEXT_5
- DBX_XREF
- DBX_WKSN_XREF

The core tables are activated when the Implement Audit/History Trail flag is set during product installation. For details, see the Product Level History File Activation section later in this chapter.

History Trail Processing-Special Considerations

The following items describe special considerations when using history processing.

- Information contained in the history tables can be queried using any SQL tool. The Repository editors do not maintain this data.
- If an object within a set is having history traced, then history is kept for all objects in the set and not individual object types. The exception to this statement is processing of items that affect the core tables.
- History collection for the text tables starts when the history flag for an object type is set to on.
- History collection for the DBX_XREF starts when the history flag for an object type is set to on. History is not collected if the only columns updated are TSO_LOGON and TIMESTAMP.
- History collection for the DBX_WKSN_XREF starts when the history flag for the Workstation Objects are set to on.
- When reviewing history information collected on text processing (DBX_TEXT_1 through DBX_TEXT_5) the entries listed in the HST_TEXT_x table show a value of D in the ZTRIGGER column if text entries are made using WsoServer and xxx. This is due to the processing of text entries results in a deletion of the text entry followed by an insertion of the new text entry. The only time that a HIST_TEXT_x table displays a value of U in the ZTIRGGER column is when Append Text is set to Y. This happens when you are using the Text Load utility (DBXTXTI).

Product-Level History File Activation

Activation of the History File at the product level occurs when the Implement Audit/History Trail Field is activated using Installation Option 2: DB2 defaults or the Installation Upgrade Customization menu. Once this field is selected and the subsequent steps for each of the process are completed, the associated history files for the core tables and triggers are created. See the steps outlined in the chapter "Installing for the First Time" or the chapter "Upgrading to the Current Release" in the *CA Repository for z/OS Installation Guide*. Activation of the History File at the product level results in the creation of history tables for core tables and their associated triggers.

The following is a list of the core tables and their associated history tables and triggers that are created during this process

Core Table Name	History Table Name	Trigger Name	Activated When Used
DBX_TEXT_1	HST_TEXT_1	DBX_TEXT_1_HSTU	Update
DBX_TEXT_1	HST_TEXT_1	DBX_TEXT_1_HSTD	Delete
DBX_TEXT_2	HST_TEXT_2	DBX_TEXT_2_HSTU	Update
DBX_TEXT_2	HST_TEXT_2	DBX_TEXT_2_HSTD	Delete
DBX_TEXT_3	HST_TEXT_3	DBX_TEXT_3_HSTU	Update
DBX_TEXT_3	HST_TEXT_3	DBX_TEXT_3_HSTD	Delete
DBX_TEXT_4	HST_TEXT_4	DBX_TEXT_4_HSTU	Update
DBX_TEXT_4	HST_TEXT_4	DBX_TEXT_4_HSTD	Delete
DBX_TEXT_5	HST_TEXT_5	DBX_TEXT_5_HSTU	Update
DBX_TEXT_5	HST_TEXT_5	DBX_TEXT_5_HSTD	Delete
DBX_XEF	HST_XREF	DBX_XREF_HSTU	Update
DBX_XEF	HST_XREF	DBX_XREF_HSTD	Delete
DBX_WKSN_XREF	HST_WKSN_XREF	DBX_WKSN_XREF_HSTU	Update
DBX_WKSN_XREF	HST_WKSN_XREF	DBX_WKSN_XREF_HSTD	Delete

Notes:

- Activation of the History flag at the product level does not activate history tracking.
- Activation of the history collection for the text tables occurs immediately after the history flag for an object type is turned on.
- Activation of the history collection for DBX_XREF occurs immediately after the history flag is turned on. History collection for DBX_XREF will not occur if the only two columns updated are TSO_LOGON and TIMESTAMP.
- Activation of history collection for DBX_WKSN_XREF occurs immediately after the history flag for any of the workstation object is turned on.

The triggers on the DBX_XREF and text tables work on an individual object type basis and not on the table level. If an object within a set is having history tracked, then history is kept for all objects in the set and not individual object types. However, for the core tables this is not true. This is because this feature looks at the entity type in the core tables; it knows to track history when that entity type has the history flag set to on (Y). The triggers on the metadata tables do not look at the entity type due to performance implications.

Activate Product-Level History File When Installing for the First Time

To activate product-level history file selection during initial installation of the product

1. Select option **2** DB2 defaults from the Installation panel.
2. Specify **Y** in the Implement Audit/History Trail field.

   The DBXPARM member of your HLQ.CUST.ISPPLIB is updated to reflect a value for the History.
3. Complete the remainder of the steps outlined in the "Installing for the First Time" chapter.

   The necessary tables and triggers are created during completion of the steps that activate Product Level History file selection.

Important! If you are installing product level History files for the first time, follow instructions in the "Installing for the First Time" chapter of the *CA Repository for z/OS Installation Guide*.

Activate Product-Level History File When Upgrading to the Current Release

To activate History file selection at the product level during an upgrade from the current release

1. Select **3: SQL** and specify **4** for the release statement to generate the core history tables and triggers on the Installation Upgrade Customization menu.
2. Select option **2: DB2 defaults** and specify **Y** in the Implement Audit/History Trail field.

   The DBXPARM member of your HLQ.CUST.ISPPLIB is updated to reflect a value for the History.
3. Complete the remainder of the steps outlined in the chapter "Installing for the First Time." The necessary tables and triggers are created during completion of these steps, which activate Product Level History file selection.

Important! If you are installing product level History files during upgrade to a current release, follow instructions in the "Upgrading to the Current Release" chapter of the *CA Repository for z/OS Installation Guide*.

Activation of Product-Level History File After Product Installation Is Complete

1. Select option **2: DB2 defaults** and specify **Y** in the Implement Audit/History Trail field.

   The DBXPARM member of your HLQ.CUST.ISPPLIB is updated to reflect a value for the History.
2. Select **3: SQL** from the Installation Upgrade Menu and specify **4** for the release statement to generate the core history tables and triggers on the Installation Upgrade Customization menu.

Metadata History Object Selection

There are two methods to turn on the history flag for a repository metadata table. The first is through the use of QUICK.MAP and the second is through the use of the EXTEND.MODEL. For details, see the following sections.

General Notes on Metadata History File Selection

The following list contains general information concerning Metadata History File Selection:

- When you specify Y in the history flag for an object and that object is in a set (multiple objects in a table), all objects in the set are audited.
- When a new object is created, the history tables and triggers are also created.
- When an existing object is recreated, the history tables and triggers are also recreated.
- If you have numerous metadata objects that you want to activate for history file collection, instead of executing a Process.BLDHST for each metadata object, you can run the Resync process. Detail information about the Resync process can be found in the "Upgrading to the Current Release" chapter of the *CA Repository for z/OS Installation Guide*.

To access the Resync process

1. Start the ARZOS installation CLIST.
2. Select option **13** - Upgrade Menu.
3. Select option **7**: Resync.

   The JCL is generated from the Resync process.
4. Submit the job-a condition code of 0 is expected.

Metadata History File Selection Using QUICK.MAP

Follow these steps to use Quick.map for metadata file selection

1. Select a map for the object about which you want to collect history.
2. Select details.metaent.
3. Specify **Y** for the value of History Flag in the Entity Details dialog.
4. Back out to the main Quick Map panel and select **Process.Save** to save the change to the object.

   A message appears stating: *Map has been saved to the control tables* successfully.

   Reply **OK**.
5. Select **Process.BLDHST.** JCL is generated that creates the history table and trigger for the selected map ID.

6. Submit the job.
7. Review the output from the job-a condition code of 0 is expected.

For more information about Quick Map, see the "Adding and Customizing Maps and Entity Types'' chapter of this guide.

Metadata History File Selection Using EXTEND.MODEL

To select metadata history file data using the Extend Model

1. Import the object for which you want to collect history
2. Go to dialog EXTEND entity type META ENT by selecting SYSTEM->EXTEND->IMPORT from the menu.
3. Select CRITERIA to define the SQL SEARCH CRITERIA, from the AR/ZOS IMPORT ENTITY LIST panel.
4. Execute the criteria by pressing **F3** or **PF3**.
5. Select the object that you want to import.
6. Specify a status into which to import the object.
7. Type **VLE** on the command line and select the object you want to import.
8. Change the value of History Flag to **Y**.
9. Select EDIT->UPDATE from the menu.
10. Select System ->Extend->Build.
11. Using Quick Map for the META ENT TYPE CODE that is defined in the object's definitions, complete all of the instructions listed in the Metadata History File Section Using QUICK.MAP earlier in this chapter.

History collection begins immediately on the Core tables. History collection will not start on the metadata table until the History table and triggers are created. This will occur when running the BLDHST command in Quick Map or running the Resync process using the install/upgrade panels.

Stop Auditing an Object Type

To stop auditing a metadata object

1. Using QUICK.MAP, select the map for the object you want to stop history collection on.
2. Select details.metaent.
3. Change the history file selection to **N**.
4. Select Process.Save to save the change to the object. A message appears stating: *Map has been saved to the control tables successfully*.

5. Reply **OK**.
6. Select Process.SAVE.
7. Select Process.REMHST.
8. JCL is generated that drops the tablespace associated with the metadata object and the associated triggers.
9. Submit the job. You will receive a condition code of 0.

Notes:

- If the history file section is changed to **N** but the Process.REMHST is not completed, history file collection resume.
- The Process.REMHST command in Quick Map generates DDL to drop the triggers on the table and the History tables. You should only execute this command if **all** objects in the set have their history flag set to off (N).

Chapter 17: Migration Extensions

This chapter describes the utilities that you can use to load and unload migration extensions:

- PRMMODL
- Migrate User Updates

This section contains the following topics:

PRMMODL Utility (see page 267)
Migrate User Updates Utility (see page 271)

PRMMODL Utility

The PRMMODL utility is used to unload user extensions and optionally, all data, from the Repository control tables.

PRMMODL creates an output data set that is a PDS. Each of the control tables is unloaded into a specific member of the output PDS. The user must specify a four-character Prefix, that goes on the front of each member of the output PDS. The member name suffix is automatically generated by the program, and denotes the control table from which the data was unloaded.

Therefore, for a user prefix of TEST and an output PDS of USERID.PRMMODL.OUTPUT produces the following output files:

DD NAME	Data Set Name (Member)	Control Table
DIALOG	USERID.PRMMODL.OUTPUT(TESTDLG)	DBX_DIALOG
MAPDESC	USERID.PRMMODL.OUTPUT(TESTMAP)	DBX_IO_MAP_DESC
MAPATTR	USERID.PRMMODL.OUTPUT(TESTMAPA)	DBX_IO_MAP_ATTR
SCRLIT	USERID.PRMMODL.OUTPUT(TESTSCRL)	DBX_SCREEN_LIT
SCRATTR	USERID.PRMMODL.OUTPUT(TESTSCRA)	DBX_SCREEN_ATTR
PRODMAP	USERID.PRMMODL.OUTPUT(TESTPRDM)	DBX_PRODUCT_MAP
CODETB	USERID.PRMMODL.OUTPUT(TESTCDT)	DBX_CODE_TBL
CODEVAR	USERID.PRMMODL.OUTPUT(TESTCDTV)	DBX_CODE_VARIANCE
MSGDESC	USERID.PRMMODL.OUTPUT(TESTMSG)	DBX_MSG_DESC
MSGTEXT	USERID.PRMMODL.OUTPUT(TESTMSGT)	DBX_MSG_TEXT
SET	USERID.PRMMODL.OUTPUT(TESTSET)	DBX_SET_TYPE
ENTDESC	USERID.PRMMODL.OUTPUT(TESTENT)	DBX_ENT_TYPE_DESC
COMMAND	USERID.PRMMODL.OUTPUT(TESTCMD)	DBX_SEC_DEF
NAVIGAT	USERID.PRMMODL.OUTPUT(TESTNAV)	DBX_NAVIGATE_DEF

DD NAME	Data Set Name (Member)	Control Table
POLICY	USERID.PRMMODL.OUTPUT(TESTPLC)	DBX_POLICY
STATUS	USERID.PRMMODL.OUTPUT(TESTSTS)	DBX_STATUS

Note: All sixteen members of the output PDS are created on every run, although some (or all) of them may be blank.

Space permitting, the output of more than one run can be added to the same PDS, as long as the Prefix is changed. If the same Prefix is re-used, the corresponding members are over-written.

The output file must be defined with an LRECL of 320 and a Record format of FB. Make sure that the file is large enough to hold a complete set of control data.

PRMMODL Utility Parameters

The PRMMODL utility is used to unload user extensions and optionally, all data, from the Repository control tables. The UNLOAD CONTROL DATA screen appears when you select **2** from the Utilities Menu.

```
------------------------- UNLOAD CONTROL DATA -------------------------
… COMMAND ===>                                                        …
…                                                                     …
… Enter unload model parameters                                       …
…                                                                     …
… ===> Debug Mode    : N       Y = Debug, N=No Debug                  …
… ===> Extension Type: C       C - Unload custom extensions only      …
…                              A - Unload all control data            …
… ===> Map Number    : 0       Map number to be unloaded.  A zero     …
…                              indicates all maps                     …
… ===> Member Prefix : TSTP    First 4 characters of unload member    …
…                              names                                  …
… ===> Output Dataset Name:                                          …
…           userid.PRMMODL.OUTPUT                                    …
…                            The Output data set must be a PDS        …
…                            that already exists with the following   …
…                            characteristics:  Lrecl 320 RECFM FB     …
…                                                                     …
… Enter ENTER command to continue or END command to EXIT.            …
```

Debug Mode

Specify Y in this field to run the job with the trace facility turned on.

Specify N (default) in this field to run the job without activating the trace facility.

Note: In general, this should only be used at the request of a Support person from CA in the event of a problem.

Extension Type Control Tables

- When C is specified only those entries in the Control Tables entered by the users unload. This is not part of the metadata supplied by CA at the last install or upgrade of the Repository product.
- Map tables unload maps where the MAP_ID is within the user-defined range (between 1000 and 2500).

- Statuses and Navigations-the CA-supplied entries are well known and the program automatically excludes them from the unload process.
- When A is specified, all user control data and CA-supplied entries are unloaded.

  Note: This is not a required field. If you do not specify a value in this field, the utility defaults to C for custom.

  Extensions are those rows that have an N in the column called PLATINUM_SUPPLIED with the following exceptions:

  - Extension Maps have a MAP_ID that is between 1000 and 25000
  - A Status that does not have the following names: DBXT, DBXP, DBXS, DB2P, DB2S, DB2T, DDLP, DDLS, DDLT, EXTP, EXTS, EXTT, IMSP, IMSS, IMST, PROD, SYSTEM, and TEST
  - All Navigations are treated as extensions

Map Number

To unload the entries from the MAP control tables for just one map by specifying its MAP_ID number on this line.

To unload **all** MAPS specify a 0 for the MAP_ID. If zero or a map number is specified, then the extension type is ignored and only the specified Map control data is unloaded.

Member Prefix

Specifies a required four-character, user-defined prefix that is used in creating the output member name that distinguishes the output members from a given run of the program.

Output Data Set Name

Identifies the input the data set name of the PDS where the various members will be generated. As the screen specifies, it must be pre-allocated and have a DCB specifying Fixed Block with a logical record of 320 characters.

To execute the PRMMODL utility

1. Select **2** from the ARZOS Utility Menu (ARZOS U).
2. Define unload the parameters; press **Enter** when you are done.

   The JCL is generated.
3. Submit the job.
4. Review the output from the job. A condition code of 0 is expected.

Migrate User Updates Utility

The MIGRATE USER UPDATES utility is used to load user extensions, and optionally, all data, back into the Repository control tables from the flat files that were previously unloaded using the PRMMODL utility. In general, it is assumed that the data is unloaded from one Repository and then loaded into the control tables of some other Repository. Hence, the term, *migrate,* to describe this process.

The unload utility, PRMMODL, creates an output data set that is a PDS. Each of the control tables described in the PRMODDL Utility section is unloaded to a member that has a predetermined suffix. The user supplies a four-character prefix.

This prefix designates which *run* of the PRMMODL utility is being stored in members with that prefix. This is how the user can distinguish between the members that were unloaded using the C (custom user extensions) option and those which were unloaded using the A (all) option. The user must also specify this Prefix (and the data set) on the Migrate User Updates Utility, and must ensure that the two *runs* are in sync. Several scenarios have been envisioned, they are:

- ADD
- REPLACE
- COPY

A description of each scenario follows.

ADD

Add adds the user extensions from one repository to the control tables of another repository. The user may want to migrate extensions (updates) from a test repository to the production repository-this is the ADD option.

If the user does not want to disturb either the existing CA-supplied control entries, or any user extensions other than the ones that are to be loaded, the user should:

1. Run the PRMMODL Utility with the C (Custom user extensions) option. If there are user extensions in the target (Prod) Repository that correspond to those that are to be loaded, (the ones that are unloaded by the PRMMODL program), they will have to be deleted from the target Repository first.
2. Run the Migrate User Updates utility, using the A option. The scripts generated by the load utility will clear those extensions from the target Repository tables using the output of PRMMODL as a guide, and then load the contents of the files into the target Repository using the resume load option.

REPLACE

The REPLACE option enables a user to migrate extensions made to a source repository to the target repository and replace the CA-supplied control data, from the latest installation.DATA.

To replace the CA-supplied metadata and leave the user extensions

1. Run the PRMMODL Utility with the C (Custom user extensions) option. Since the load of the CA-supplied control data uses a replace type of load that wipes out the target table first, it is necessary to unload the user extensions using PRMMODL with the C option, and then load them back in after the CA metadata is refreshed.
2. Run the Migrate User Updates Utility, using the R option. The script generated by the load utility clears the target table first and then load the CA metadata, much the same way as a re-install of the product would do.

   In particular, it uses the Replace option of the DB2 load utility, so the target table is cleared. The CA utility then loads the contents of the PRMMODL output members using the Resume option.

COPY

A user should use this option when they want to transfer, or copy, the control information, both CA-supplied and user extensions, from one Repository to another. The Copy option copies the entire content of the control tables of one repository to another. The target control tables are exact copies of the source control tables.

Note: The target repository control tables are cleared before the new data is copied in. Again, the source control data must be unloaded using PRMMODL (with the A option) as previously described.

To transfer, or copy, the control information, both CA-supplied and user extensions, from one Repository to another

1. Run the PRMMODL utility with the A (All) option. The A option unloads everything from the control table into the PRMMODL output member.

   Note: If you use this option, care must be taken that the output PDS is allocated with sufficient space.

2. Run the Migrate User Updates utility, using the C option. The C option of the CA utility loads the output member of the PRMMODL to the Target Repository control table using the Replace option of the DB2 load utility.

   Note: Care must be taken that the PRMMODL output member involved was generated using the A option of PRMMODL.

   Running the COPY option of the Migrate User Updates utility with the output members generated with the C option of PRMMODL results in a repository that has only the user's extensions and none of the CA-supplied control data.

The Migrate User Extension Parameters

The Migrate User Updates utility loads user extensions and optionally, all data, back into the Repository control tables from the flat files that were previously unloaded using the PRMMODL utility.

The parameters you can specify in this Migrate User Extension utility are show in the following sample of the Migrate User Update dialog.

```
--------------------------- LOAD USER EXTENSIONS ----------------------
COMMAND ===>
Enter Model Extension Load parms
Load control tables...
DIALOGs? ===> Y CODE TABLEs? ===> N
ENTITYs? ===> N IO MAPs? ===> Y
MESSAGEs? ===> N COMMANDs? ===> N
NAVIGATIONs? ===> N STATUSs? ===> N
Type of Load? ===> A (A - Add, R - Replace, C - Copy)
===> Member Prefix : TEST First 4 characters of unload member
names
===> Output Dataset Name:
userid.PRMMODL.OUTPUT
The Output data set & member prefix
used in running PRMMODL
Enter ENTER command to continue or END command to EXIT.
```

Load Control tables

Unlike PRMMODL, which unloads all tables all the time (except when a single MAP_ID is selected), the Migrate User Updates utility enables the user to select individual groups of tables to load separately. The user determines which groups of tables are needed to support given extensions. Obviously, an extension that created new entities must also create new maps, and both must be loaded-at least eventually, for the extension to work. However, the utility gives the user the flexibility to load groups of tables separately if the user wishes to.

The groups (some of which are single tables) are:

DIALOGs?	DBX_DIALOG DBX_PRODUCT_MAP
CODE TABLEs?	DBX_CODE_TBL DBX_CODE_VARIANCE
ENTITYs?	DBX_ENT_TYPE_DESC DBX_SET_TYPE DBX_POLICY
IO MAPs?	DBX_IO_MAP_ATTR DBX_IO_MAP_DESC DBX_SCREEN_LIT DBX_SCREEN_ATTR
MESSAGEs?	DBX_MSG_DESC DBX_MSG_TEXT
COMMANDs?	DBX_SEC_DEF
NAVIGATIONs?	DBX_NAVIGATE_DEF
STATUSs?	DBX_STATUS

Type of Load

(ADD, (REPLACE, (COPY - described previously.

Member Prefix

Specifies the four-character prefix of the PRMMODL output PDS members that you want the utility to use for this load.

Output Dataset Name

Specifies the **PRMMODL** output PDS the members of which you want the utility to load.

To run migrate user extensions

1. Run the PRMMODL utility to unload the control data to be loaded. For details, see the PRMMODL section earlier in this chapter.

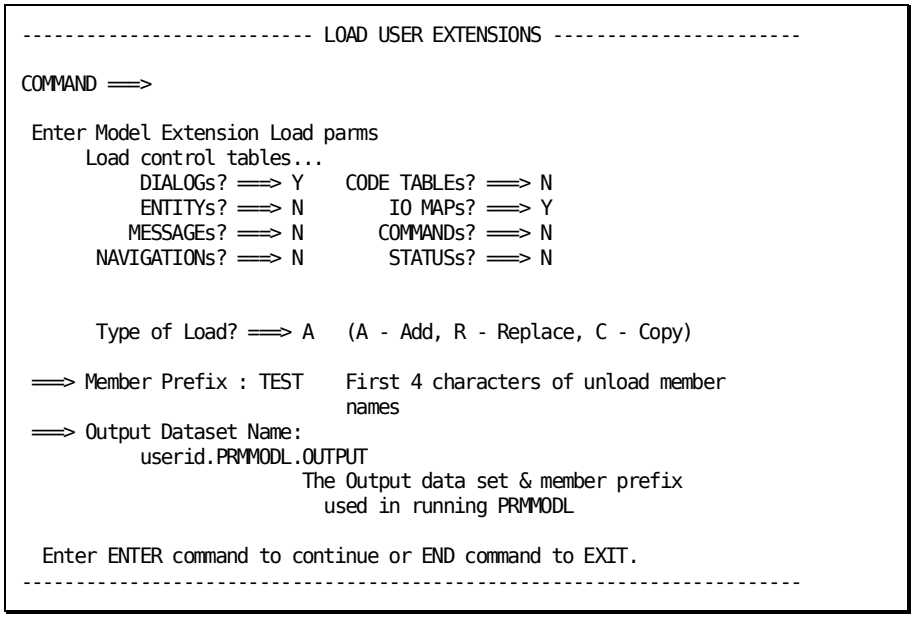
   - When you specify C (custom user extensions) as the extension type in PRMMODL, you will specify either A (Add) or R (Replace) as the Type of Load when executing the Migrate User Updates Utility.
   - When you specify A (all) as the extension type in PRMMODL, you will specify the C (COPY) as the Type of Load when executing the Migrate User Updates Utility.

   The PREFIX specified for the members or the data set name selected for the output PDS must be used to distinguish which option was used for the unload and subsequent load.

   Note: It is up to the user to ensure that the data set name and member prefix used in this load correspond to the appropriate unload from PRMMODL.

2. To run the Migrate User Updates utility, select **9** from the ARZOS Utilities Menu. The Load User Extensions screen appears. A sample screen follows.

   The following is a sample Load User Extensions screen.



```
--------------------------- LOAD USER EXTENSIONS ----------------------

COMMAND ===>

 Enter Model Extension Load parms
      Load control tables...
           DIALOGs? ===> Y    CODE TABLEs? ===> N
           ENTITYs? ===> N        IO MAPs? ===> Y
         MESSAGEs? ===> N       COMMANDs? ===> N
       NAVIGATIONs? ===> N        STATUSs? ===> N


       Type of Load? ===> A   (A - Add, R - Replace, C - Copy)

 ===> Member Prefix : TEST    First 4 characters of unload member
                              names
 ===> Output Dataset Name:
           userid.PRMMODL.OUTPUT
                          The Output data set & member prefix
                            used in running PRMMODL

  Enter ENTER command to continue or END command to EXIT.
-------------------------------------------------------------------------
```

3. Define the load control tables.
4. Define the type of load.
5. Define the member prefix.
6. Define the output data set name.

7. Press Enter; the JCL is generated.
8. Submit the job. The condition code should not exceed 4.

Chapter 18: Creating CA Repository Paths

CA Repository paths are collections of steps that progress from one meta-entity type to another through the associations or relationships that connect the types. These paths are defined by the administrator and used extensively by the Repository Path Report, Path Delete, and Path Workstation Add facilities.

When you retrieve a root occurrence from the repository and execute one of the facilities that use paths, the chosen path processes all the occurrences that are connected to the root occurrence. The DBX_XREF table is used exclusively to determine each branch. The data tables are not accessed until the report is ready to be printed.

This chapter explains the repository paths and how to define these paths.

This section contains the following topics:

Path Report Facility (see page 277)
Path Delete Facility (see page 278)
Path Workstation Add Facility (see page 279)
How Paths Are Processed (see page 281)
Load Paths (see page 281)
Add or Modify the Paths Definitions (see page 281)
Create Path Attribute Statements (see page 287)
Specify the Path Facility Parameters (see page 289)
Path Facilities Usage Notes (see page 291)

Path Report Facility

The Path Report Facility:

- Provides impact analysis reports that reach beyond the standard XREF Report
- Includes definitions indirectly linked to the root occurrence
- Uses paths that string together multiple associations and relationships
- Contains every entity type encountered along the path, which simplifies the process of determining which definitions are linked together (when there is no direct connection between them).

The paths used by Path Reports:

- Direct the facility from one definition to another along established routes
- Specify which attributes of those definitions should be displayed in the reports

If no attributes are explicitly included in the report, then only the key attributes will be reported.

Execute the Path Report Facility

To execute the Path Report Facility

1. From CA Repository, choose VIEW.DIALOG to access a dialog such as the DB2 dialog.
2. Choose the VIEW.TYPE to access an entity type such as TABLE.
3. Execute OPTIONS.REPORTS.PATH; this displays a list of all available path names.
4. Select an applicable Path such as DB2ALL by typing 'S' next to the path name.
5. Press Enter to display the generated JCL.

Be aware of the following:

- If no attributes are explicitly included in the report, then only the key attributes will be reported.
- Not all entity types have paths that are supplied by CA Repository.

For more information about how to generate a report, see the *CA Repository for z/OS Product Guide*.

Path Delete Facility

Use the Path Delete Facility to delete definitions linked to one another along a pre-defined path.

Execute the Path Delete Facility

The Path Delete Facility can only be executed in batch mode, as follows:

1. Use EDIT, SPECIAL, PATHDEL to generate the JCL that executes this facility; this displays a list of all available path names.
2. Select an applicable path that can process the root object, then press Enter to bring up the generated JCL.

The Delete flag determines when the definition encountered along a path should be deleted from the repository.

You can specify that the definition is to be deleted if it does not participate as a:

- Source of any other relationships
- Target of any other relationships
- Both a Target and Source of any other relationship

See the definition of the Delete flag in the Path Facility Parameters section for more information.

Run the Path Delete Facility in Report Mode

It is possible to run the Path Delete Facility in report mode. Report mode updates the repository, prints the report, and then issues a rollback. Use Report mode when testing new Path Delete statements.

Be aware of the following:

- All objects deleted from the repository are displayed on the report.
- Relationships and associations found through the Via Type are not printed in the report, as their deletion automatically occurs in the IO module when a From or To Type gets deleted.

See the Checkpoint parameter in the Path Facility Parameters section of the manual for more information.

Path Workstation Add Facility

Workstations are conceptual groupings used in locking, migrations, and with CASE interfaces. Workstations contain occurrences of any and all meta-entities.

The Path Workstation Add Facility:

- Adds groups of related repository definitions to a repository workstation
- Uses paths that string together multiple associations and relationships

Objects Added to the Workstation

Be aware of the following:

- The root object is always added to the workstation.
- Related entities and relationships are added depending on the specification of the Include flag.
- Associations and relationships linking the related entities and relationships are added depending on the specification of the Print Via flag.

See the definitions of Include and Print Via flags in the Path Facility Parameters section later in this chapter for more information.

Execute the Path Workstation Add Facility

The Path Workstation Add Facility can only be executed in batch mode, as follows:

1. Choose EDIT, SPECIAL, PATHADD to generate the JCL that executes this Facility. A list of all available path names is displayed.
2. Select an applicable path that can process the root object. A list of workstations is then displayed.
3. Select the workstation to which the objects will be added, then press Enter to bring up the generated JCL.

Run the Path Add Facility in Report Mode

It is possible to run the Path Add Facility in report mode. Report mode will update the repository, print the report, and then issue a rollback. Use Report mode when testing new Path Add statements.

Note:

- All objects added to the Workstation are displayed on the report.
- Associations that are added to the workstation do not contain key fields so only the Entity name of the association is displayed.

See the Checkpoint parameter in the Path Facility Parameters section of the manual for more information.

How Paths Are Processed

The Path Report Facility, Path Delete Facility, and Path Workstation Add Facility all process paths by:

- Traversing the path
- Collecting all of the metadata definitions encountered along the path
- Processing those definitions in the order in which they are encountered

Load Paths

In order to load a path, you must generate the JCL to submit the load procedure.

To load a path

1. Generate the JCL to submit the load procedure as follows:
   a. Select the Path Load option from the CA Repository Installation Panel. (Option 17 from the main installation menu.)Execute the DB2 Load Utility to load new path definitions.
   b. The DBX_PATH table contains the Path statements. The Path Add, Path Delete and Path Report options use these statements to determine how to navigate through the repository.

      The DBX_PATH_ATTRIBUTE table contains the attributes that will be displayed when executing a Path Report.
2. Customize the JCL to meet your site specifications.
3. Add to or modify the paths definitions in the SYSRC00 files of both steps. See the section that follows for instructions about how to do this.
4. Submit the job.

Be aware of the following: If the SYSRC00 file does not include every path desired by the site, modify the control statements for the DB2 load utility to specify RESUME instead of REPLACE to load additional rows into the tables.

Add or Modify the Paths Definitions

Note: Changes to the CA Repository maps, including creation of new maps, modifications to attributes, and changes to existing sources and targets, may necessitate modification to the CA Repository paths. Be sure that you understand the impact of map changes upon existing paths before you undertake the modifications. See the Specifying the Path Facility Parameters for information about PARM//DIALOG.

To define a path you must:

- Create Path definition statements
- Create Path attribute statements

Create Path Definition Statements

The following table describes the format and the information contained in the path definition statements.

Column	Description
Path Name	A one to eight character name for the Path All statements with the identical Path Name constitute a single path.
From Type	The Entity Type of the Root Node in the Path statement For example, to create a Path statement that will traverse from TABLE to ELEMENT through the COLUMN relationship, the From Type would be 126. The From Type determines the order in which the entity types encountered along the path are processed. The step of the path with the From Type that corresponds to the entity type of the root occurrences will be executed first. The next step that will be executed will be the one whose From Type matches the To Type or Via Type of the root occurrence's step. This continues until there are no steps of this path whose From Type matches the previous To Type or Via Type. Note: The From Type cannot be a set.
Via Type	The entity type of the relationship or association that links the two objects together For example to create a Path statement that will traverse from TABLE to ELEMENT through the COLUMN relationship, the Via Type would be 102. **Note:** The Via Type cannot be a set.
To Type	The entity type of the child node in the Path statement. For example to create a Path statement that will traverse from TABLE to ELEMENT through the COLUMN relationship, the Via Type would be 204. **Note**: The To Type can be a SET. Also, the To Type cannot be an association.
Include flag	Enter Y to have the To Type displayed in the report, included in the workstation path, or deleted from the repository Enter N if the To Type is not to be displayed in the report, added to the workstation, or deleted from the workstation. For simple one-line paths, the To Type should be included in

Column	Description
	the report. For more complex paths much of this information may be unnecessary. For instance to begin at an ELEMENT and end with ENTITYTPs linked to that ELEMENT, the path must contain two statements: ELEMENT - ATTR EL - ATTR TP and ATTR TP - AT DE ET - ENTITYTP. By specifying INCLUDE= 'Y' for the first statement, the report will show the Logical attribute types that link the physical elements to logical entity definitions. By specifying INCLUDE='N' on the first step, the report will suppress the ATTR TPs and instead show a direct link between the physical elements and logical entity definitions. **Note:** When adding objects to a workstation, the Include flag works independently from the Print Via flag. It is possible to add the To Type object to the workstation without adding the Via Type object.
Direction Flag	The Direction is used to determine if the From Type is a Source or Target of the Via Type. Enter: D-If the From Type is a Source of the Via Type U-If the From Type is a Target of the Via Type For example, to create a Path statement that will traverse from TABLE to ELEMENT through the COLUMN relationship, the Direction flag would be D.
Print Via flag	Determines whether the Via Type should be printed in the report or added to the workstation The Print Via flag must be set to Y if the Via Type is to be used as the From Type in another statement. For instance, to display 88-levels for a particular REC MAP you would need to have at least two path statements such as RECORD -- REC MAP - ELEMENTS and REC MAP - RM VALUE - VALUE. For the Path Report to process RM VALUEs, the Print Via flag on the first statement must be specified as Y. **Note:** When adding objects to a workstation, the Print Via flag works independently from the Include flag. It is possible to add the Via Type object to the workstation without adding the To Type object. Also, when running a Path Report the Via Type will not be printed if the To Type is not included.
Display Sequence	Determines the sequence of events when one particular entity type is used as the From Type in more than one step in the same path definition When running a Path Report, a statement with a sequence of 1 will be displayed before a statement with a sequence of 2. When running a Path Delete, occurrences connected through paths with lower sequences will be deleted before those connected through higher sequences.

Column	Description
	Note: The values of this field do not have to be sequential or start with the value of 1.
Delete Rule	Determines which entities along a path will be deleted when running a Path Delete The Delete rule applies to the To Type. The valid values for Delete Rule are: S-Indicates that the To Type will not be deleted if the object is used as a Source of any relationship or association. T-Indicates that the To Type will not be deleted if the object is used as a Target of any relationship or association. B-Indicates that the To Type will not be deleted if the object is used as either a Source or Target of any relationship or association. A-Indicates that the To Type should always be deleted. Blank-Indicates that the To Type will not be deleted. **Note:** The root object is always deleted.
Path Description	A one to forty character description of the path Every statement in the path must have the same description.

Example of a Path Definition Statement for a Path Report

The following table contains the data that would be entered to define a path report for COPYBOOK.

#	Path Name	From Type	Via Type	To Type	In	Dir	Prt Via	Disp Seq	Del Rule	Description
1	COPYBOOK	557	189	153	Y	D	Y	0		COPYBOOK PATH
2	COPYBOOK	557	570	200	Y	D	Y	0		COPYBOOK PATH
3	COPYBOOK	557	26010	557	Y	D	Y	0		COPYBOOK PATH
4	COPYBOOK	153	26012	26010	Y	D	Y	0		COPYBOOK PATH
5	COPYBOOK	200	26012	26010	Y	D	Y	0		COPYBOOK PATH
6	COPYBOOK	200	25434	216	Y	D	Y	0		COPYBOOK PATH

#	Path Name	From Type	Via Type	To Type	In	Dir	Prt Via	Disp Seq	Del Rule	Description
7	COPYBOOK	153	25434	216	Y	D	Y	0		COPYBOOK PATH
8	COPYBOOK	200	203	209	Y	D	Y	0		COPYBOOK PATH
9	COPYBOOK	153	203	209	Y	D	Y	0		COPYBOOK PATH
10	COPYBOOK	200	210	205	Y	D	Y	0		COPYBOOK PATH
11	COPYBOOK	153	210	205	Y	D	Y	0		COPYBOOK PATH
12	COPYBOOK	25434	26011	26010	Y	D	Y	0		COPYBOOK PATH
13	COPYBOOK	25434	25439	205	Y	D	N	0		COPYBOOK PATH
14	COPYBOOK	25434	25438	209	Y	D	N	0		COPYBOOK PATH

Path Report Processing

The information below describes the processing that will occur as each path statement in the previous table is executed.

The root of the path will be printed. In the previous example, COPYBOOK is the root.

1. For each COPYBOOK print both the COPY SEG relationship and the target SEGMENT entity.
2. For each COPYBOOK print both the COPY REC relationship and the target RECORD entity.
3. For each COPYBOOK print both the COPYCOPY relationship and the target COPYBOOK entity.
4. For each SEGMENT print both the RS COPY relationship and the target COPYCOPY relationship.
5. For each RECORD print both the RS COPY relationship and the target COPYCOPY relationship.
6. For each RECORD print both the REC MAP relationship and the target object in the SUB ELS set.

7. For each SEGMENT print both the REC MAP relationship and the target object in the SUB ELS set.
8. For each RECORD print both the EL VALUE relationship and the target VALUE.
9. For each SEGMENT print both the EL VALUE relationship and the target VALUE.
10. For each RECORD print both the FUNCTION relationship and the target object in the ELEMENTS set.
11. For each SEGMENT print both the FUNCTION relationship and the target object in the ELEMENTS set.
12. For each REC MAP relationship print both the RM COPY relationship and the target COPYCOPY relationship.
13. For each REC MAP print only the target object in the ELEMENTS set which is the target of the RM FUNC relationship.
14. For each REC MAP print only the VALUE entity that is the target of the RM VALUE relationship.

Example of a Path Delete

The following table contains the data that would be entered for a path definition statement for a path delete.

#	Path Name	From Type	Via Type	To Type	In	Dir	Prt Via	Disp Seq	Del Rule	Description
1	JOBS	256	263	257	Y	D	N	0	A	JOBS PATH
2	JOBS	257	264	151	Y	D	N	0	B	JOBS PATH
3	JOBS	257	25106	259	Y	D	Y	0	B	JOBS PATH
4	JOBS	25106	25105	270	Y	D	N	0	S	JOBS PATH
5	JOBS	257	25103	25102	Y	D	Y	0	B	JOBS PATH
6	JOBS	25103	25104	270	Y	D	N	0	S	JOBS PATH

Delete Path Processing

The information below describes the processing that will occur as each path statement in the previous table is executed.

The root of the path will be deleted. In the previous example, JOB is the root.

1. For each JOB always delete the JOB STEPs.
2. For each JOB STEP, navigate to the PROGRAM through the STEP PRG association and delete the PROGRAM if the PROGRAM is neither a source nor target of any other relationship or association.
3. For each JOB STEP, navigate to the FILE through the STEPFILE relationship and delete the FILE if the FILE is neither a source nor target of any other relationship or association.
4. For each STEPFILE relationship, navigate to the DATA FIL through the FILE DS relationship and delete the FILE DS if it is not a source of any other relationship or association.
5. For each JOB STEP, navigate to the PROC through the STEPPROC relationship and delete the PROC if the PROC is neither a source nor target of any other relationship or association.
6. For each STEPPROC relationship, navigate to the DATA FIL through the STPPRCDS relationship and delete the FILE DS if it is not a source of any other relationship or association.

Create Path Attribute Statements

The following table describes the format and information contained in the path attribute statements.

Note: Path Attributes are not used when running a Path Add or Path Delete.

Column	Description
Path Name	A one to eight character name for the Path Each statement in the Path must have the same name. The Path Name must also match the Path Name in the Path Definition Statements.
From Type	The entity type of the Root Node in the Path statement When the value of the From Type is the same as the value in the To Type then the specified attribute will be displayed only when the Path Report encounters and displays an occurrence of the From/To Type as the root occurrence.
Via Type	The entity type of the relationship or association that links the two

Column	Description
	objects together When the Via Type is the same as both the From Type and the To Type, then the specified attribute will be printed for the Via Type when the Path Report encounters and displays an occurrence of the Via Type. **Note:** You must specify Print Via = 'Y' on the Path definition statement to print these attributes.
To Type	The entity type of the child node in the Path statement When the Path Report encounters and displays an occurrence of the To Type by navigating the Via Type, then the specified attributes will be displayed in the report. The Value in From Type is unimportant and can be any value except the value in the To Type (which would indicate a root occurrence).
Attribute Name	The name of the column in the underlying DB2 table that will be displayed for the Via Type or To Type The prefix "S." or "T." can be used to indicate that the attribute belongs to the Source or Target types of the relationship found in the Via Type or To Type.
Comment	A comment about the attribute

Example of a Path Attribute Statement for a Path Report

The following table contains the data that would be entered to define path attributes for the COPYBOOK path report.

#	Path Name	From Type	Via Type	To Type	Attribute	Description
1	COPYBOOK	557	557	557	LANGUAGE	ROOT ATTRIBUTES
2	COPYBOOK	557	570	200	INCLUDE_01	COPYREC FROM COPYBK
3	COPYBOOK	557	570	200	PREFIX	COPYREC FROM COPYBK
4	COPYBOOK	557	570	200	SUFFIX	COPYREC FROM COPYBK
5	COPYBOOK	200	25434	204	DATA_TYPE	ELEMENT FROM RECMAP
6	COPYBOOK	200	25434	204	LENGTH	ELEMENT FROM RECMAP
7	COPYBOOK	200	25434	204	DECIMAL	ELEMENT FROM RECMAP
8	COPYBOOK	25434	25434	25434	FILLER_LENGTH	RECMAP ATTRIBUTES
9	COPYBOOK	25434	25434	25434	VALUE	RECMAP ATTRIBUTES

Path Attribute Processing

The information below describes the processing that will occur as each path statement attribute in the previous table is encountered.

1. Print the LANGUAGE attribute when processing the Path Report for a COPYBOOK.
2. Pint the INCLUDE_01 flag from COPY REC when processing a record through the COPY REC relationship.
3. Print the PREFIX from COPY REC when processing a RECORD through the COPY REC relationship.
4. Print the SUFFIX from COPY REC when processing a RECORD through the COPY REC relationship.
5. Print the DATA_TYPE from ELEMENT when processing an ELEMENT through the REC MAP relationship.
6. Print the LENGTH from ELEMENT when processing an ELEMENT through the REC MAP relationship.
7. Print the DECIMAL from ELEMENT when processing an ELEMENT through the REC MAP relationship.
8. Print the FILLER_LENGTH when processing and printing a REC MAP.
9. Print the VALUE when processing and printing a REC MAP.

Specify the Path Facility Parameters

The JCL that executes the Path Report, Path Delete and Path Add statements are almost identical. The only difference between the three JCL jobs is the parameters used when calling the DBXBATP program.

The format of the PARM= statement on the EXEC card in the JCL is shown below.

```
PARM=('/SUBS.TYPE.USAGE')
```

where:

SUBS	Specifies the DB2 subsystem name in which the Platinum Repository has been installed
TYPE	Specifies the type of Path Facility being run. Valid Values are PATH - Path Report PATHADD - Path Add PATHDEL - Path Delete

Run-Time Parameters for the SYSIN Data Card

A second set of parameters is found in the SYSIN data set. The format of these parameters is described in the following table.

Parameter Line	Description
PARM//CHKPNT=n	Specifies when commits are performed when running Path Deletes or Path Adds Valid values are: N - Commits are performed after the last delete when running a Path Delete. Commits are performed at the end after all workstation adds are complete when running a Path Add. Y - Commits are performed after each delete when running a Path Delete. Commits are performed at the end after all workstation adds are complete when running a Path Add. R - Commits are not performed when running a Path Add or Path Delete. Instead after the report is printed, a rollback occurs. This option will help facilitate testing of Path Deletes and Path Adds. Note: If Y is specified when running a Path Delete and an interruption occurs in the process, a restart of the Path Delete may not be possible as some objects may have been deleted and those deletes were committed. Instead you should specify N when running a Path Delete. Also, any value can be specified when running a Path Report, as the checkpoint parameter is not used.
PARM//WKSN= workstation name	Specifies the name of the workstation to which the Path Add will add occurrences of objects encountered along the specified path. The WKSN parameter is not used when running Path Deletes or Path Reports
PARM//DIALOG= dialog	Specifies the name of the dialog to use. The Dialog determines which maps to use If a root occurrence Via Type or To Type is encountered and that type is not found in the dialog, the Key Name, Status and Version are printed using the first map found in the control tables for that Type. When printing non-key attributes as defined in the Path Attribute table, the attributes must be found in the map specified by the dialog.
PARM//PATH=path name	Specifies the name of the Path to be used by the Path Report, Path Delete and Path Add Facilities
type//name//status//version	Specifies the Entity Name, Key Name, Status and Version of the root occurrence selected by the user **Note:** Multiple root occurrences can be specified.

Example of the SYSIN Data Set

```
//SYSIN DD *
PARM//CHECKPT=N
PARM//WKSN=Path Add Workstation
PARM//DIALOG=SYSTEM
PARM//PATH=JOBS
JOB//DBXSYNC/DBXT//00
/*
```

Path Facilities Usage Notes

Be aware of the following:

- The From Type and Via Type cannot be a set. Instead create additional statements for each object within the set.
- The Path Facility cannot be run from a relationship or association. The Root object must be an Entity.
- At times the last node off the root object in the report will be indented incorrectly. This usually occurs if the last node has subordinate nodes.
- The Path Add Facility and the Path Report Facility will not print key information about an association; instead just the name of the entity type is displayed.
- Rows within the Path Report are sorted on ENT_ID and cannot be sorted in any other order due to the processing of the DBX_XREF table.

Chapter 19: Using the Relational Translator

This chapter explains how to use the Relational Translator to create physical DB2 model definitions.

This section contains the following topics:

The Relational Translator (see page 293)
Open the Relational Translator Screen (see page 296)
View the Results of Processing (see page 299)
The Work Tables (see page 299)
Control Table Layouts (see page 304)

The Relational Translator

Use the Relational Translator to create a physical RDBMS model using relational model definitions.

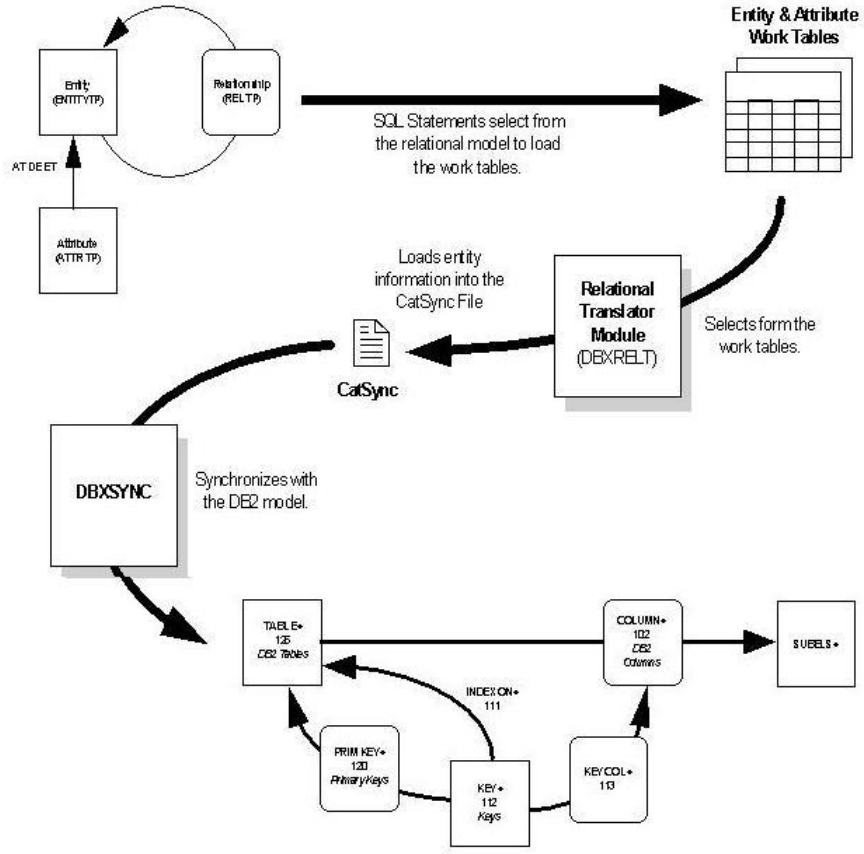
The Relational Translator process translates logical entities, relationships and attributes to create table, column, index, primary key, and foreign key definitions. Once translated, this data can be used for generation DDL using the CA Repository.

Note: The Relational Translator can also be executed in a test mode. This allows you to determine in advance how existing Repository data would be affected by a translation.

The Relational Translator is a batch process that is submitted from the repository. The process involves three basic steps:

1. Loading data from a relational or logical model into work tables.
2. Creating an input file for the Catalog Synchronization utility.
3. Running the catalog synchronization process.

The following diagram shows this process. A portion of the DB2 model referenced by the Relational Translator is shown in the lower right corner.

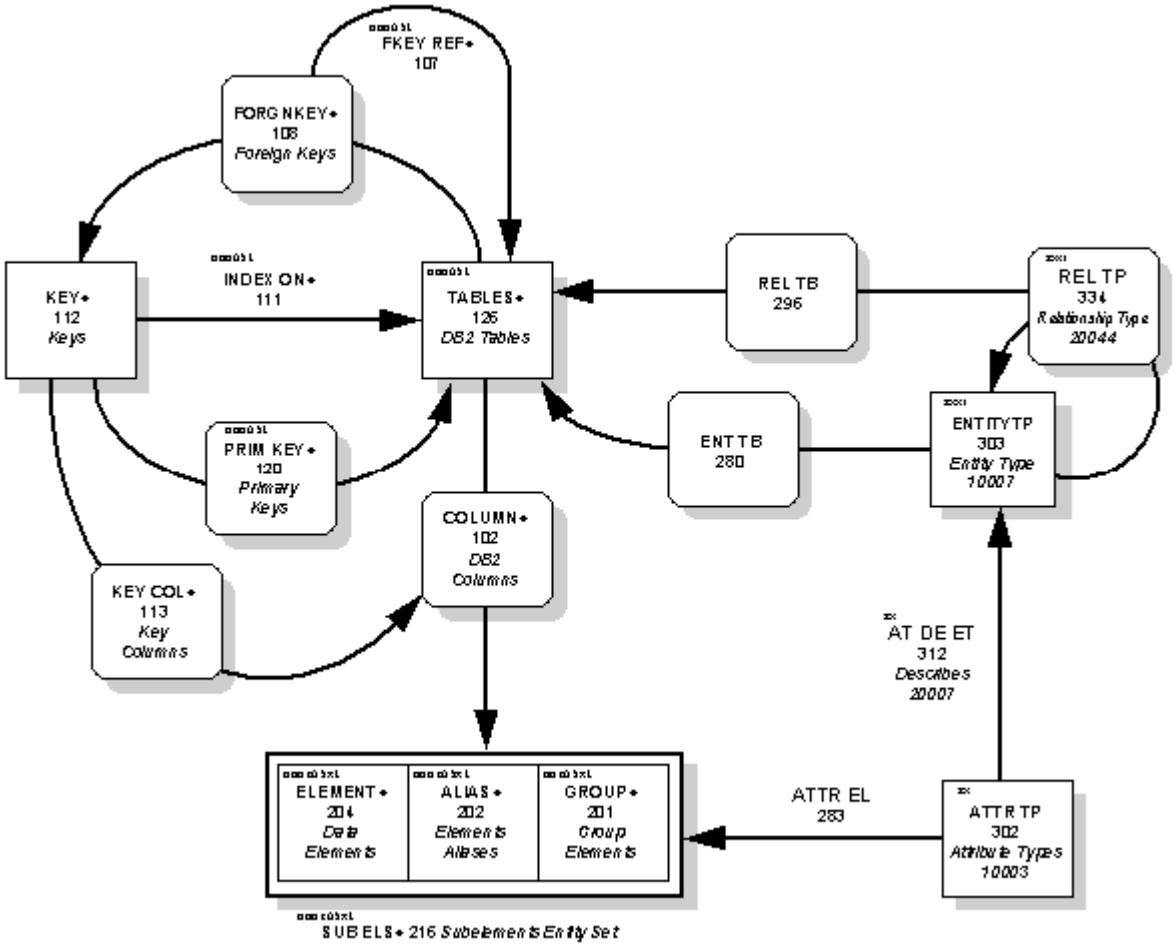


The Relational Transfer Process

The logical models loaded into the work tables are selected from one of the repository relational models. The work tables contain all the information required to create definitions of tables (TABLE), columns (COLUMN), indexes (KEY), primary keys (PRIM KEY), foreign keys (FORGNKEY), and base table views (VIEW). An option to default physical storage options is provided.

Tie-Back Cross Referencing

If the information loaded into the work tables comes from one of the repository relational models, you can create tie-back associations for this new information. In the following diagram, the associations ENT TB, REL TB and ATTR EL are used to tie the relational to the physical.



Workstation Processing

Use Workstation IDs to:

- Select logical data assigned to a Repository workstation
- Assign the new DB2 entities to the same or different workstation.

You are asked to specify a From workstation and a To workstation. If you do not specify a To workstation, the new data is added to the From workstation.

Entity and Attribute Name Generation

Entity and attribute names are easily extended beyond the maximum name length for DBMS objects (for example, DB2's 18 character limit for tables, columns and indexes). Since the length of element names used by most programming languages is less then the names allowed for attributes, the Relational Translator uses the Name Generation utility to create meaningful and syntactically correct physical names. The correct delimiters are used for both DBMS objects (_) and COBOL element names (-).

Element Re-Usability

Before creating new entities as part of a translation, the repository checks up to three statuses and one version to find existing elements that match the incoming data. If no match is found in the first status, the second and third statuses are searched. If no match is found in any of the three statuses, a new element is inserted. You specify the statuses and version searched by the Repository before processing.

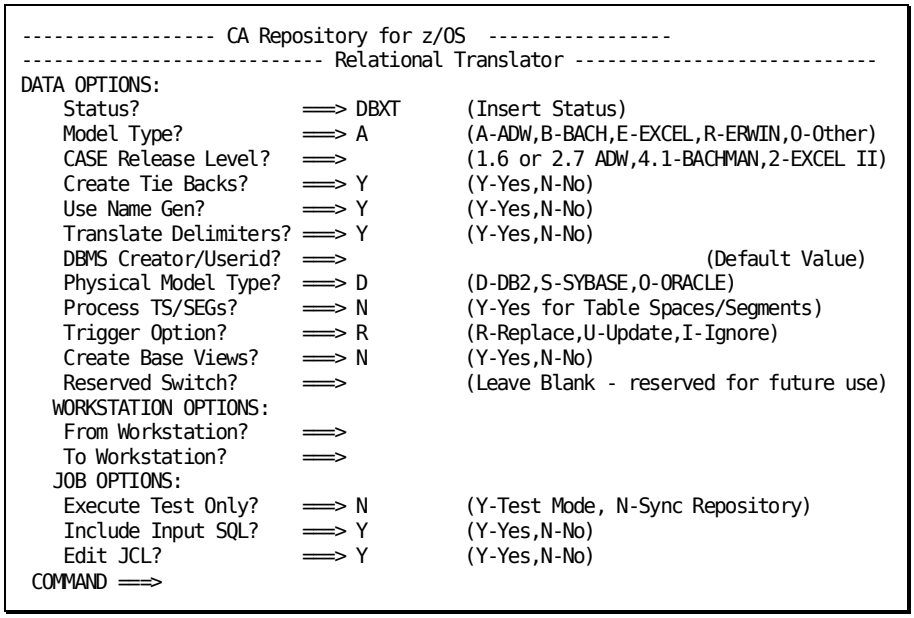
Open the Relational Translator Screen

The repository provides a screen driven facility for submitting the relational translation process. Use the following procedure to open the Relational Translator screen.

To open the Relational Translator screen

1. From the CA Repository, choose VIEW, DIALOG; this displays the Extend dialog.
2. Choose VIEW, TYPE to access any entity type; an Edit screen for the entity type you select is displayed.

3. Execute OPTIONS, RELTRAN. The Relational Translator screen appears as shown below.



```
------------------ CA Repository for z/OS  -----------------
--------------------------- Relational Translator ---------------------------
DATA OPTIONS:
    Status?               ===> DBXT      (Insert Status)
    Model Type?           ===> A         (A-ADW,B-BACH,E-EXCEL,R-ERWIN,O-Other)
    CASE Release Level?   ===>           (1.6 or 2.7 ADW,4.1-BACHMAN,2-EXCEL II)
    Create Tie Backs?     ===> Y         (Y-Yes,N-No)
    Use Name Gen?         ===> Y         (Y-Yes,N-No)
    Translate Delimiters? ===> Y         (Y-Yes,N-No)
    DBMS Creator/Userid?  ===>                              (Default Value)
    Physical Model Type?  ===> D         (D-DB2,S-SYBASE,O-ORACLE)
    Process TS/SEGs?      ===> N         (Y-Yes for Table Spaces/Segments)
    Trigger Option?       ===> R         (R-Replace,U-Update,I-Ignore)
    Create Base Views?    ===> N         (Y-Yes,N-No)
    Reserved Switch?      ===>           (Leave Blank - reserved for future use)
   WORKSTATION OPTIONS:
    From Workstation?     ===>
    To Workstation?       ===>
   JOB OPTIONS:
    Execute Test Only?    ===> N         (Y-Test Mode, N-Sync Repository)
    Include Input SQL?    ===> Y         (Y-Yes,N-No)
    Edit JCL?             ===> Y         (Y-Yes,N-No)
 COMMAND ===>
```

4. Enter values into the Relational Translator input fields as follows:

Field	Enter
Status	The status to be used for any new inserts performed by the Relational Translator process.
Model Type	A-ADW B-BACHMAN E-Excelerator R-ERwin O-Other
CASE Release Level	1.6 for ADW 2.7 for ADW 4.1 for Bachman 2 for Excelerator II This field is not used when the ERwin or other model type is chosen.
Create Tie Backs	Y-To create associations that link entities and relationships to tables (ENT TB and REL TB/PSETTABL) and attributes to elements (ATTR EL) N-To ignore this option
Use Name	Y-To use the name generation option for generating TABLE,

Field	Enter
Gen	KEY, COLUMN and ELEMENT names N-If you do not want to use the name generation option for generating TABLE, KEY, COLUMN and ELEMENT names If you specify N, the first 18 characters of the entity or attribute name are used to create the name.
Translate Delimiters	Y-If you want existing elements to be searched for using the different delimiter types N-If you do not want existing elements to be searched for using the different delimiter types If an element is not found, underscores in the name are changed to dashes and the translator searches for the element again. If the element is still not found, dashes are changed to spaces and the translator searches again.
DBMS Creator ID	The creator ID to be used for all new tables and indexes. If left blank, no creator is included for these entity types.
Physical Model Type	D-DB2 S-SYBASE O-ORACLE
Process TS/SEGs	Y-To generate the physical objects (table spaces or segments) required for the model N-If you do not want to generate the physical objects (table spaces/segments) required for the model If you specify Y, default storage groups (DB2, ORACLE), table spaces (DB2, ORACLE), index spaces (DB2), or segments (SYBASE) are inserted into is provided by editing the JCL.
Trigger Option	R-Replace; add trigger information, replacing any existing information) U-Update; add trigger information, appending it to any existing information I-Ignore; ignore/do not add trigger information
Create Base Views	Y-To create definitions for base table views for any table definitions inserted by the Relational Translator N-If you do not want to create definitions for base table views for any table definitions inserted by the relational translator
From Workstation	The workstation ID that entities and relationships must be assigned to in order to be selected into the work tables
To Workstation	The workstation ID that all new inserts are assigned to. If the workstation specified does not exist, a new one will be created.
Execute Test Only	Y-To perform a test translation

Field	Enter
	N-To perform a repository synchronization. The test mode will not perform any entity insertions or update the Repository. However, the output of the test job lists all the actions that would have been necessary to perform a real synchronization.
Include Input SQL	Y-To include the set of insert statements that are used to populate the work tables for a specified workstation ID. N-If you do not want to include the set of insert statements that are used to populate the work tables for a specified workstation ID If you specify N, the information currently in the work tables is used.

5. After you complete all of the fields, press Enter to start processing.
6. Submit the JCL by typing SUB on the command line.

View the Results of Processing

During the catalog synchronization portion of the relational translator process, changes made to the repository are written to the Sync Report table.

To print these actions, execute the OPTIONS, REPORTS, CATSYNC command. For more information about this report, see the *CA Repository RDBMS* Administration *Guide*.

Use the test option and the CatSync report to determine the effects that translating a particular relational model will have on your existing Repository data.

The Work Tables

The relational translator uses the Entity and Attribute control tables to temporarily store all entity, relationship, and attribute information. These tables are explained in the following sections.

The Entity Control Table

The DBX_REL_TRAN_ENT contains ENTITY and RELATIONSHIP data that is translated into TABLE and FORGNKEY entities. Descriptions of each column and how it is used are provided in the table below.

Column	Description
ENT_ID	An integer field used to store the internal ID of the entity or relationship
TSO_LOGON	A character field used to store the TSO LOGON ID of the person loading data into the Relational Translator work tables
WORKSTATION_ID	A character field used to define the From workstation ID
TYPE	A character field used to indicate either entity (ET) data, relationship data (RT) or foreign key data (FK)
NAME	A character field used to store the name of the entity or relationship
STATUS	A character field used to store the status of the entity or relationship
VERSION	A small init field used to store the version of the entity or relationship.
CREATE_BY	A character field that specifies the create date of the entity or relationship; reserved for future use
CREATE_ DATE	A date field that specifies the create ID of the entity or relationship. Reserved for future use
MOD_BY	A character field that specifies the last update ID of the entity or relationship; reserved for future use
MOD_DATE	A date field that specifies the last update date of the entity or relationship; reserved for future use
MOD_TIME	A time field that specifies the last update time of the entity or relationship; reserved for future use
DESCRIPTION	A character field used to store the description of the entity or relationship

The Attribute Control Table

The table DBX_REL_TRAN_ATTR contains attribute data that is translated into ELEMENT, COLUMN, KEY COL data. In addition, PRIM KEY, INDEX ON and FKEY REF associations are derived from this information. Descriptions of each column and how it is used are provided in the tables that follow.

Column	Description
ENT_ID	An integer field used to store the internal ID of the attribute

Column	Description
TSO_LOGON	A character field used to store the TSO LOGON ID of the person loading data into the Relational Translator work tables
WORKSTATION _ID	A character field used to define the From workstation ID for the entity or relationship associated with the attribute
SOURCE_ENTITY_ID	An integer field used to store the ID of the entity or relationship that points to this attribute
FK_SOURCE_ID	An integer field used to store the ID of the entity that is converted to a table having a foreign key (FORGNKEY)
FK_REF_ID	An integer field used to store the ID of entity that is converted to a table and is referenced by a foreign key
COLUMN_ID	An integer field used to store the column ID for this attribute
TYPE	A character field used to indicate either entity attribute data (EA) data, relationship attribute data (RA) or primary key columns (PK)
NAME	A character field used to store the name of the attribute
STATUS	A character field used to store the status of the attribute
VERSION	A smallinit field used to store the version of the attribute relationship
SEQ_NUM	A character field used to order the attributes of an entity type
LENGTH	A character field used to specify the length of an attribute field
DECIMAL	A character field used to specify the decimal positions of an attribute field
DATA_TYPE	A character field used to specify the data type of the attribute
NULLS	A character field used to specify the nulls rule for an attribute
CREATE_BY	A character field that specifies the creator ID of the attribute; reserved for future use
CREATE_ DATE	A date field that specifies the create ID of the attribute; reserved for future use
MOD_BY	A character field that specifies the last update ID of the attribute; reserved for future use
MOD_DATE	A date field that specifies the last update date of the attribute; reserved for future use
MOD_TIME	A time field that specifies the last update time of the attribute; reserved for future use
DESCRIPTION	A character field used to store the description of the attribute

Model Requirements

The relational translator is a generic facility designed to translate logical Entity-Relationship models into CA Repository DB2 objects.

The input model can be one of several provided by different CA Repository products or the standard E-R model included in the LOGICAL dialog. In addition, most non-standard E-R models that have been created in the CA Repository can be used as input to the relational translator. This section describes what is required by the Relational Translator to create valid DB2 tables, indexes, primary keys and foreign keys.

As mentioned before, the relational translator process is a combination of three steps:

1. The first steps insert into the work tables.
2. The next step creates the new DBMS definitions by processing the work table information.

   Data is loaded into the work tables by a series of SQL statements that insert into the work tables using data selected from repository storage tables. The select statements are displayed as part of the JCL and can be modified to fit a site's E-R model. A sample statement that selects ENTITYTP occurrences for the workstation IEW PROD is shown in the JCL in the next two figures. CA Repository Technical Support is available to assist sites if customization to the supplied SQL statements is required.
3. The last step loads the newly defined physical model into the Repository using the Catalog synchronization process.

For the Entity-Relationship models provided as part of the CA Repository, there are several rules that determine the creation of different DBMS components. These rules are discussed in the following section.

The Logical Dialog Model to Create DBMS Component

DBMS component creation considerations:

- The ENTITYTP type in the LOGICAL dialog translates to TABLE entities in the DBMS dialog.

  If the name generation option is used, the table's name is generated from the ENTITYTP name, otherwise the name is the first 18 characters of the ENTITYTP name, depending on the physical target DBMS selected. The table's CREATOR ID is supplied by the user in the Relational Translator screen.

- All associated ATTR TP entities through the AT DE ET association represent columns on the table.

  The physical attributes of the column and element (DATA_ TYPE, LENGTH, DECIMAL and NULLS) are represented by the same fields on the ATTR TP. If the ATTR TP is assigned to an ELEMENT via the ATTR DT association, the DATA_TYPE, LENGTH and DECIMAL values of the elements will be used.

- A primary key is created for every table created.

  The KEY entity is given the same name as the table. Primary key columns are defined by those attributes that have the values MIN = 1, MAX = 1 and MAX PER VALUE = 1. If no attributes are found that meet these criteria, a default ELEMENT, COLUMN, and KEY COL will be added using the name of the ENTITYTP.

The REL TP type in the LOGICAL dialog determines how foreign keys are created. The following rules are used when selecting from the REL TP entity type:

- If both the FROM-TO MAXIMUM (RELA_LR_MAX) and the TO-FROM MAXIMUM (RELA_RL_MAX) have values not equal to 1, the relationship represents a table. The columns of the table are the primary keys of the two entities (ENTITYTP) that are being related. Foreign keys are created between the new relationship table and the tables that were intended to be related in the logical model.
- If either the FROM-TO MAXIMUM (RELA_LR_MAX) or the TO-FROM MAXIMUM (RELA_RL_MAX) have values equal to 1, the relationship represents one or two foreign keys (FORGNKEY) between the source entity and the target entity. The following scenarios are possible:
  - If FROM-TO MAXIMUM = 1 AND TO-FROM MAXIMUM Ö 1, a foreign key referencing the target entity's table will be created on the source entity's table. The target table's primary key will be added to the source table as another index.
  - If FROM-TO MAXIMUM Ö 1 and the TO-FROM MAXIMUM = 1, a foreign key referencing the source entity's table will be created on the target entity's table. The source table's primary key will be added to the target table as another index.
  - If FROM-TO MAXIMUM = 1 and the TO-FROM MAXIMUM = 1, a foreign key is created on both the source and target entities' tables. Each foreign key references the other entity's table. The primary keys of each table are added to the other table as an additional index.

Control Table Layouts

The following tables contain layout information for each of the relational translator control tables.

DBX_REL_TRAN_ATTR

Name	Description	Data Type
ENT_ID	CA Repository internal identifier for entities/relationships	INTEGER
TSO_LOGON	ID of individual loading data	CHAR (8)
WORKSTATION_ID	FROM workstation ID	CHAR (15)
SOURCE_ENTITY_ID	ID of entity/relationship pointing to this attribute	INTEGER
FK_SOURCE_ID	Foreign Key source ID	INTEGER

Name	Description	Data Type
FK_REF_ID	ID of entity reference by this foreign key	INTEGER
COLUMN_ID	Column ID for this attribute	INTEGER
TYPE	Indicates entity (ET) relationship (RT) or foreign key (FK)	CHAR (2)
NAME	CA Repository entity name	CHAR (75)
STATUS	CA Repository status	CHAR (8)
VERSION	CA Repository version	CHAR (6)
SEQ_NUM	Sequence number	CHAR (32)
LENGTH	Length of the attribute field	CHAR (4)
DECIMAL	Decimal positions of the attribute field	CHAR (4)
DATA_TYPE	Data type of the attribute	CHAR (8)
PICTURE_CLAUSE	Picture clause of the attribute	CHAR (32)
NULLS	NULLS rule of the attribute	CHAR (1)
CREATE_BY	Created by this user	CHAR (8)
CREATE_DATE	Created on this date	DATE
CREATE_TIME	Create at this time	TIME
MOD_BY	Last modified by this user	CHAR (8)
MOD_DATE	Last modified on this date	DATE
MOD_TIME	Last modified at this time	TIME
DESCRIPTION	Additional information about the entity	VARCHAR (240)

DBX_REL_TRAN_ENT

Name	Description	Data Type
ENT_ID	CA Repository internal identifier for entities/relationships	INTEGER
TSO_LOGON	ID of individual loading data	CHAR (8)
WORKSTATION_ID	FROM workstation ID	CHAR (15)
TYPE	Indicates entity (ET), relationship (RT), or foreign key (FK)	CHAR (2)
NAME	CA Repository entity name	CHAR (75)

Name	Description	Data Type
STATUS	CA Repository status	CHAR (8)
VERSION	CA Repository version	CHAR (6)
CREATE_BY	Created by this user	CHAR (8)
CREATE_DATE	Created on this date	DATE
CREATE_TIME	Create at this time	TIME
MOD_BY	Last modified by this user	CHAR (8)
MOD_DATE	Last modified on this date	DATE
MOD_TIME	Last modified at this time	TIME
DESCRIPTION	Additional information about the entity	VARCHAR (240)

Chapter 20: Working with User Exits

This chapter explains how to add and write user exits.

This section contains the following topics:

User Exits (see page 307)
Add User Exits (see page 307)
Writing an Exit (see page 309)
List of ISPF Variables Available in User Exits (Shared Pool) (see page 310)
Install User Exits (see page 313)
Restore User Exits (see page 318)

User Exits

User exits:

- Provide additional security
- Enforce naming standards
- Provide additional types of validation

Examples of user exits include the following commands:

- OPTIONS.NAME
- OPTIONS.OFFSET
- OPTIONS.RECALC

Add User Exits

You can add user exits to existing commands or to entirely new user-created commands.

To add a user exit

1. From CA Repository, choose VIEW, DIALOG to access the Extend dialog.
2. Choose VIEW, TYPE to access the COMMAND entity type. The Command Edit screen displays.

```
SIZE ----------- CURRENT DIALOG: EXTEND   ENTITY TYPE: COMMAND ----------MAX
-      FILE EDIT VIEW OPTIONS SYSTEM PROFILE NAVIGATE HELP                  -
-                 LAST ACTION: NOTHING+                                1 OF 1
-       HEADER INFORMATION:
-        WINDOW COMMAND          ===>
-        STATUS                  ===>
-        VERSION NUMBER          ===>    (0 - 32767)
-        DESCRIPTION             ===>
-                                ===>
-       COMMAND DEFINITION:
-        WINDOW                  ===>
-        FUNCTION                ===>
-        SUB FUNCTION            ===>
```

```
-        COMMAND                 ===>
-        ACTION BAR SEQUENCE     ===> 0     (1 - 32767)
-       COMMAND EXIT PROCEDURE:
-        PROCEDURE NAME          ===>
-        PROCEDURE TYPE          ===>   (C-CLIST,P-PANEL,G-PROGRAM)
-        EXECUTION TIME          ===>
-
------------------------------------------------------------------------------
```

3. Enter values into the COMMAND EXIT PROCEDURE input fields, as follows:

Field	Enter
PROCEDURE NAME	The name you want to give to your exit.
PROCEDURE TYPE	C-CLIST or Rexx Exec P-Panel G-Program
EXECUTION TIME	A-After B-Before

4. Use EDIT.INSERT to update the Repository.
5. Use the SYSTEM.EXTEND.BUILD command to re-build the command entity. The Repository calls the exit before and after the command is executed (determined by the EXIT EXECUTION TIME attribute).

Note: Exits can only be added to commands under the OPTIONS or EDIT functions of the Edit window. If you add an exit to the DELETE, INSERT, or UPDATE commands, you must also add exits to the SYNC command. The SYNCACT ISPF variable is used to ensure the exit is only invoked with the appropriate action. Depending on the action required by the current entity, set SYNCACT to I, U, or D when you execute the SYNC command.

Writing an Exit

You can write exits as CLISTS, Panels, or Programs. The repository communicates with these exits using the ISPF shared pool. Some commands included with the repository are implemented using exits and can be used as examples when creating new exits. These commands/exits and the variables they use are described in the following table:

Facility or Command	Purpose
COPYBOOK Import Facility	An exit used to submit jobs from the repository This facility uses a CLIST, Panel, and skeleton to perform file tailoring and job submission.
OFFSET command	A command used to calculate the correct offset for MAP ATTR relationships This command is implemented via a CLIST. The CLIST is called once for each tagged entity on the screen and it calculates the offset based upon the previous offset and length. The ISPF variables OFFSET, ATTRLEN, and ATTRCODE are defined on the map for the relationship to provide the necessary information for the CLIST.
RECALC command	A command used to calculate the starting position of fields in an IMS SEGMENT automatically It is implemented using a CLIST which calls a program. The program calls the I/O module to calculate the length (in bytes) of the element to determine the starting positions. The CLIST is called once for each tagged entity on the edit screen.

Note: Exits can call the I/O module, and can use DB2. Care must be taken when defining the maps to be used by the exits to ensure that any attributes needed by the exit have ISPF variable names defined for them.

List of ISPF Variables Available in User Exits (Shared Pool)

The following table lists ISPF Variables and their description.

ISPF Variable	Description
BJC1	First line of batch job card
BJC2	Second line of batch job card
BJC3	Third line of batch job card
BJC4	Fourth line of batch job card
TAPE	CA Repository tape volume serial
TUNIT	Unit for the tape
EXPDT	Expiration date for tape
UNIT	Unit for data sets
DASD	DASD volume serial
TDSNQ	Prefix for CA Repository data sets
REPOWNER	CA Repository DB2 table qualifier
CATOWNER	DB2 Catalog table qualifier.
PRMPLAN	Repository plan
PRMPCKC	Repository collection name
COMPANY	Company name
LPP	Lines per page for reports
DATABASE	Default DB2 database used for DBEXCEL objects
STOGROUP	Default storage group used for CA Repository objects
SUBS	DB2 subsystem where the CA Repository is installed
LOADLIB	CA Repository load library
DBRMLIB	CA Repository DBRMLIB
DB2LOAD	DB2 load library for DSNTIAD
DB2DBRM	DB2 DBRMLIB for DSNTIAD
DB2LD2	DB2 load library
DB2DBR2	DB2 DBRMLIB
SQLID	User ID for SET SQLID in DB2 create and security steps
DB2REL	DB2 release for CA Repository subsystem

ISPF Variable	Description
DB2VER	DB2 version for CA Repository subsystem
MAXLIST	Default maximum number in list
TEXT1	Name of first text set
TEXT2	Name of second text set
TEXT3	Name of third text set
TEXT4	Name of fourth text set
TEXT5	Name of fifth text set
BPARM1 to BPARM5	Values specified for the site batch parameters
TIADPLN	DSNTIAD plan name
BPOOL	Buffer pool for Repository tablespaces
IBPOOL	Buffer pool for Repository indexes
WLMNAME	Work Load manager region name

Variables Put Out by Repository for Each Entity Before Executing the Exit

The following table lists Repository variables for each entity before executing the exit.

ISPF Variable	Description
SRCID	CA Repository source entity ID
TGTID	CA Repository target ID
ENTTYPE	CA Repository internal identifier for the entity type
SRCTYPE	CA Repository internal identifier for the source type
TGTTYPE	CA Repository internal identifier for the target type
ENTNUM	Number of the current entity (i.e. the number on the left of the horizontal screen for this entity)
ENTNAME	CA Repository name for the entity
DBXFUNC	Function for the current command
DBXSUBFN	Subfunction for the current command
DBXCMD	Current command
DBXDLG	Current dialog

ISPF Variable	Description
DBXLOGON	TSO ID for the current user
DBXPARM	Current parm added to the command
FIRSTSW	Switch set to *Y* if this is the first time the exit is called for this command
IOKEY	Current valid CA Repository key to call the I/O routine
SYNCACT	A special variable for the EDIT.SYNC command denoting whether the current entity is being Inserted
SSNM1	Name of the subsystem currently attached to
PLAN1	Name of the plan currently attached under
DEBUG	Current debugging status (Y or N)
Plus, any other variables defined by the ISPF name for attributes.	

Variables Retrieved by Repository After Executing the Exit

The following table lists Repository variables retrieved after executing the exit.

ISPF Variable	Description
DBXRETC	Return code from the exit if not equal to 0
DBXMSGQ	Qualifier of the message to be displayed if DBXRETC not 0
DBXMSGI	ID of the message to be displayed if DBXRETC not 0
DBXREAS	Reason for the message if DBXRETC not 0
Plus, any other variables defined by the ISPF name for attributes.	

Install User Exits

The installation of CA Repository user exits is performed and controlled under SMP/E. The user exit code is created, compiled, and linked outside of SMP/E control. The object module is then received, and applied to SMP/E as a USERMOD.

To Install a User Exit

1. Create/Determine Source Code.

   You can either create your own source code or use some of the sample programs such as COBOLIO that are provided with the product. Sample source code is located in the CAI.SHLQ.SAMPPGM data set. It is recommended that you create a non SMP/E controlled dataset such as CAI.SHLQ.CUST.SAMPPGM to contol your source code.

2. Compile and Link the Source Code.

   Compile and link the source code. The resulting object module needs to reside in a user-defined data set. The data set attributes for this data set need to be DSORG=PO,RECFM=U,BLKSIZE=6144. The recommended compile options are:

   ```
   PARM=('TRUNC(BIN),NODYNAM,MAP,XREF,LIST,NOADV',          X
   'LIB,OBJECT,RENT,RES,APOST,NOSSR,NOCOMPILE(S)')
   ```

   The recommended link options are:

   ```
   PARM='LIST,MAP,NCAL'
   ```

   The following is a sample of the JCL that can be used to complete this task:

   ```
   //COMPILE EXEC  PGM=IGYCRCTL,
   //   PARM=('TRUNC(BIN),NODYNAM,MAP,XREF,LIST,NOADV',
   //        'LIB,OBJECT,RENT,RES,APOST,NOSSR,NOCOMPILE(S)')
   //STEPLIB  DD  DSN=IGY.SIGYCOMP,DISP=SHR
   //SYSLIB   DD  DSN=CAI.SHLQ.SAMPPGM,DISP=SHR
   //SYSPRINT DD  SYSOUT=*
   //SYSIN    DD  DISP=SHR,DSN=CAI.SHLQ.SAMPPGM(COBOLIO)
   //SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
   //SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
   //SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
   //SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
   //SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
   //SYSUT6   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
   //SYSUT7   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
   //SYSLIN   DD  DSN=&OBJ(COBOLIO),DISP=(NEW,PASS),UNIT=SYSDA,
   ```

```
//         DCB=(BLKSIZE=6160,LRECL=80,DSORG=PO),SPACE=(TRK,(30,10,5))
//*
//LINK  EXEC PGM=IEWL,PARM='LIST,MAP,NCAL'
//SYSPRINT DD SYSOUT=*
//SYSLMOD  DD DISP=SHR,DSN=USERNAME.EXITS
//OBJLIB   DD DISP=(SHR,PASS),DSN=&OBJ
//SYSUT1   DD UNIT=VIO,SPACE=(TRK,15)
//SYSLIN   DD  DDNAME=SYSIN
//SYSIN    DD  *
 INCLUDE OBJLIB(COBOLIO)
 ENTRY   COBOLIO
 NAME    COBOLIO(R)
/*
```

3. SMP/E RECEIVE and APPLY the USER Exit.

   The following is a sample of the JCL that can be used to SMP/E:

```
RECEIVE and APPLY the user exit.
The JCL contains comments needed for customizing.
//P$70UEXT  JOB (ACCT,INFO),'PROGRAMMER',
//     CLASS=0,MSGCLASS=A,MSGLEVEL=(1,1),
//     GROUP=,USER=*UID,PASSWORD=*PSW
//*--------------------------------------------------------------------
//*
//*      CA REPOSITORY FOR Z/OS PRODUCT
//*   * * *  ARZOS SMP/E RECEIVE & APPLY A USER EXIT.
//*
//*  EDIT AND SUBMIT THE FOLLOWING PROC SAMPLE JCL TO EXECUTE GIMSMP
//* PROGRAM TO SMP/E RECEIVE THE ARZOS PRODUCT
//*          EXPECTED CONDITION CODE EQ 00
//*
//*--------------------------------------------------------------------
//*
//***  PROCEDURE SMPEXIT FOR SMP/E RECEIVE/APPLY  OF A USER EXIT
//*
//* INSTRUCTIONS:
//*
//*   1. PRIOR TO PROCESSING RECEIVING YOUR USER EXIT IN SMP/E,
//*      THE FOLLOWING ITEMS MUST HAVE BEEN COMPLETED:
//*
//*      1A. COMPILE AND LINK THE SOURCE CODE. THE RESULTING OBJECT
//*          MODULE NEEDS TO RESIDE IN A USER DEFINED DATA SET.
//*          THE DATA SET ATTRIBUTES FOR THIS DATA SET NEED TO BE
//*          DSORG=PO,RECFM=U,BLKSIZE=6144.
//*
//*      1B. THE BASE AR/ZOS FMID, CP$7000, MUST BE APPLIED PRIOR
//*          TO SUBMISSION OF THIS JOB.
//*
```

```
//*   2. CUSTOMIZE THE JCL:
//*
//*      2A. CHANGE JOB CARD AS NECESSARY FOR YOUR SITE.
//*      2B. UPDATE THE 'P$SHLQ' SYMBOLIC TO REFLECT THE
//*          NAME OF SMPCSI WHERE THE CA REPOSITORY
//*          PRODUCT RESIDES.
//*      2C. UPDATE THE 'USERDSN' SYMBOLIC TO REFLECT THE
//*          DSN WHERE YOUR USER EXIT OBJECT MODULE RESIDES.
//*      2D. UPDATE THE DSN ON THE STEPLIB DD STATEMENT
//*          TO REFLECT YOUR SMP/E LIBRARY.
//*      2E. UPDATE THE '++USERMOD' STATEMENT TO REFLECT A
//*          UNIQUE USERMOD NAME.
//*          WE WOULD SUGGEST USING A NAMING CONVENTION THAT WILL
//*          PROVIDE YOU WITH A WAY OF EASILY IDENTIFYING ALL
//*          OF THE CA REPOSITORY USER EXITS.
//*          AN EXAMPLE OF SUCH A NAMING CONVENTION COULD BE:
//*
//*          'UP$NNNN'
//*               WHERE:
//*                      'U' INDICATES A USER EXIT
//*                      'P$' IS A CONSTANT INDICATING THAT THE
//*                           EXIT IS USED BY THE CA
//*                           REPOSITORY PRODUCT
//*                  AND
//*                      'NNNN' IS ANY USER VARIABLE INDICATING
//*                           THE UNIQUE EXIT NAME.
//*
//*          2E1. IF YOU PERFORM A GLOBAL CHANGE ON THE VALUE OF
//*               'UP$NNNN' THE VALUES ON THE '++USERMOD',
//*               'RECEIVE SELECT' AND 'APPLY SELECT' WILL ALSO
//*               BE UPDATED.
//*               OTHERWISE IT WILL BE NECESSARY FOR YOU TO ALSO
//*               UPDATE THE 'RECEIVE SELECT' AND 'APPLY SELECT'
//*               STATEMENTS MANUALLY.
//*      2F. CHANGE THE VALUE OF 'YYYYJJJ' ON THE 'REWORK'
//*          PARAMETER OF THE '++USERMOD' STATEMENT TO REFLECT
//*          THE CURRENT JULIAN DATE.
//*      2G. IN THE ++JCLIN SECTION UPDATE THE FOLLOWING:
//*          2G1. THE VALUE OF 'AAAAAAA' ON THE FIRST
//*               'INCLUDE CP$70LLD (AAAAAAA)' STATEMENT WILL
//*               NEED TO BE UPDATED TO REFLECT YOUR USER EXIT
//*               NAME.
//*          2G2. UPDATE THE 'ENTRY BBBBBBB' TO REFLECT THE
//*               THE CORRECT ENTRY NAME.
//*          2G3. UPDATE THE 'NAME   CCCCCCC(R)' TO REFLECT
//*               THE CORRECT USER EXIT NAME.
//*               BE SURE TO CHECK THE CONTENTS OF THE 'CAI.SHLQ.
//*               LOADLIB' TO VERIFY THAT THE USER EXIT NAME THAT
//*               YOU HAVE SELECTED IS UNIQUE.
```

```
//*          2G4. UPDATE THE '++MOD (DDDDDDD)' TO REFLECT THE
//*               CORRECT USER MODULE NAME.
//*          2G5. IF YOU ARE NOT USING THE DEFAULT TARGET NAME OF
//*               'P$TGT', CHANGE THE NAME ON THE 'SET BDY'
//*               STATEMENT TO REFLECT THE APPROPIRATE TARGET NAME.
//*
//*
//*      3. SUBMIT THE JOB. A CONDITION CODE OF 0 IS EXPECTED.
//*
//*      4. ONCE THE EXIT HAS BEEN SUCCESSFULLY INSTALLED USING
//*         SMP, IT WILL BE NECESSARY TO DEFINE THE EXIT TO
//*         THE CA REPOSITORY PRODUCT. REFER TO THE
//*         ADDING USER EXITS SECTION OF CHAPTER 17: WORKING WITH
//*         USER EXITS IN THE r7.2 ADMINISTRATOR'S GUIDE FOR DETAILS
//*--------------------------------------------------------------------
//*
//SMPEXIT PROC P$SHLQ='CAI.SHLQ',
//            USERDSN='USEREXIT',
//            SYSOUT='*'
//*
//SMPEXIT EXEC PGM=GIMSMP,REGION=4096K,PARM='DATE=U'
//STEPLIB  DD DSN=IBM.SMPE.LOADLIB,DISP=SHR
//SMPCSI   DD  DSN=&P$SHLQ..SMPCSI.CSI,
//         DISP=SHR
//SMPLOG   DD  DUMMY
//SMPHOLD  DD  DUMMY
//SMPOUT   DD  SYSOUT=&SYSOUT
//SMPRPT   DD  SYSOUT=&SYSOUT
//SMPLIST  DD  SYSOUT=&SYSOUT
//SMPSNAP  DD  SYSOUT=&SYSOUT
//SYSPRINT DD  SYSOUT=&SYSOUT
//USREXIT  DD DSN=&USERDSN,
//            DISP=SHR
//SMPWRK6  DD UNIT=SYSDA,SPACE=(CYL,(50,10,25)),DCB=BLKSIZE=3120
//  PEND
//*
//STEP1  EXEC SMPEEXIT
//*
//SMPEXIT.SMPPTFIN DD DATA,DLM='||'
++ USERMOD (UP$NNNN)
   REWORK(YYYYJJJ) .
++ VER (Z038) FMID(CP$7000).
++ JCLIN ASM(PGM=ASMBLR) LKED(PGM=IEWL).
//LINKIT EXEC PGM=IEWL
//*
//*     LINK EDIT THE USER EXIT
//*
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=&SYSUT1,UNIT=SYSDA,SPACE=(1024,(100,10))
```

```
//SYSLMOD  DD DISP=SHR,
//            DSN=P$SHLQ..CAILIB
//SYSLIN   DD DDNAME=SYSIN
//USREXIT  DD DSN=&USERDSN,
//            DISP=SHR
//CP$70LLD DD DISP=SHR,
//            DSN=P$SHLQ..CP$70LLD
//SYSIN    DD *
 INCLUDE CP$70LLD(AAAAAAA)
 INCLUDE CP$70LLD(DBXVLI2)
 INCLUDE CP$70LLD(DBXVLIO)
 INCLUDE CP$70LLD(DBXMSGS)
 INCLUDE CP$70LLD(DBXPCMD)
 INCLUDE CP$70LLD(DBXMCUA)
 INCLUDE CP$70LLD(DBXCODV)
 INCLUDE CP$70LLD(DBXMOVE)
 INCLUDE CP$70LLD(DBXIO)
 INCLUDE CP$70LLD(DBXEXIT)
 INCLUDE CP$70LLD(DBXMAP)
 INCLUDE CP$70LLD(DBXVCMD)
 INCLUDE CP$70LLD(DBXTRIM)
 INCLUDE CP$70LLD(DBXNUM)
 INCLUDE CP$70LLD(DBXDYNM)
 INCLUDE CP$70LLD(DBXTBIG)
 INCLUDE CP$70LLD(DBXSIOS)
 INCLUDE CP$70LLD(DBXHCMD)
 ENTRY   BBBBBBB
 NAME    CCCCCCC(R)
++MOD(DDDDDDD) DISTLIB(CP$70LLD) LKLIB(USREXIT) .
||
//SMPEXIT.SMPCNTL  DD *
  SET BDY(GLOBAL).
  RECEIVE SELECT ( UP$NNNN ) SYSMODS .
  SET BDY(P$TGT).
  APPLY   SELECT ( UP$NNNN ) REDO .
/*
```

4. Define the User Exit to CA Repository.

   Once the exit has been successfully installed using SMP/E, it will be necessary to define the exit to the CA Repository product. For more information, see the section "Adding User Exits" in this chapter.

Restore User Exits

To restore user exits previously applied to a CA Repository SMPCSI

1. SMP/E RESTORE the User Exit.
2. Customize SMP/E RESTORE JCL.
3. Change the P$SHLQ symbolic to reflect the CA Repository SMPCSI library.
4. Update the RESTORE SELECT statement, RESTORE SELECT(UP$NNNN), to reflect the User SYSMOD name that you want to restore.

   The following is a sample of the JCL that can be used to SMP/E RESTORE the User Exit, if needed.

```
//SMPEXIT  PROC P$SHLQ='CAI.SHLQ',        <== Update
//             SYSOUT='*'
//*
//SMPEEXIT EXEC PGM=GIMSMP,REGION=4096K,PARM='DATE=U'
//SMPCSI   DD  DSN=&P$SHLQ..SMPCSI.CSI,
//         DISP=SHR
//SMPLOG   DD  DUMMY
//SMPHOLD  DD  DUMMY
//SMPOUT   DD  SYSOUT=&SYSOUT
//SMPRPT   DD  SYSOUT=&SYSOUT
//SMPLIST  DD  SYSOUT=&SYSOUT
//SMPSNAP  DD  SYSOUT=&SYSOUT
//SYSPRINT DD  SYSOUT=&SYSOUT
//CAILIB   DD  DISP=SHR,
//             DSN=&P$SHLQ..LOADLIB
//  PEND
//*
//*    EXECUTE SMPE RESTORE A USER EXIT
//*
//STEP6  EXEC SMPEXIT
//*
//SMPEEXIT.SMPCNTL  DD *
       SET BDY(P$TGT).
       RESTORE SELECT(UP$NNNN).     <== Update
/*
```

5. Remove the User Exit from CA Repository for z/OS.

   Once the exit is successfully removed from SMP/E, you must remove the exit from the CA Repository for z/OS product. The procedure for removing an exit defined to CA Repository for z/OS is similar to the procedure for adding user exits. For details, see the section ”Adding User Exits”. Erase the previous values entered followed by EDIT.UPDATE then SYSTEM.EXTEND.BUILD.

Chapter 21: Approval Process

The Approval table is used to handle the approval process for Webstation Option users. This process includes:

- Putting any transactions of the Repository objects, Insert, Delete, or Update into the Approval Queue for the same or different Approvers
- The ability for a Approver to allow or reject transactions
- Inform the user about Approval status
- Creating Approval report

This section contains the following topics:

Enable the Approval Feature (see page 319)
Activate an Approval Process (see page 320)
Add Approval for an Object (see page 320)
Stop Approval Processing for an Object (see page 321)
Setting Up Approvers (see page 321)
Approving an Object (see page 322)
Approval Table (see page 322)

Enable the Approval Feature

Updates, inserts, and deletes from Webstation Option can be deferred until approval is granted.

The approval process is optional and must be explicitly set to function.

To indicate that the approval process is active you must set the Require Approval Process flag to Y on the DB2 Default's panel as part of the Installation which will update the DBXPARM file.

Once the flag is set you must execute file statement 10 to update the DBX_System Table.

Note: For more information, see the C*A Repository for z/OS Installation Guide*.

Activate an Approval Process

Once the approval process is active for the Repository, the Approval process can be implemented for individual object types.

- Use Quick maps to set the approval required flag to Y on specific objects
- Set up approvers using the Steward Object

More Information:

See "Add Approval for an Object" Selection in this guide.

Add Approval for an Object

Use QUICK.MAP to set the Approval flag for a repository metadata file selection.

To set an object for approval processing follow these steps.

1. Select a map for the object.
2. Select details.metaent.
3. Specify Y for the value of Approval Required Flag in the Entity Details dialog.
4. Press PF3 to go back to the main Quick Map panel.
   a. Select Process.
5. Click Save to save the change to the object.

   A message appears stating: Map has been saved to the control tables successfully.
   a. Select OK.

More Information:

See the "Adding and Customizing Maps and Entity Types'' chapter of this guide.

Stop Approval Processing for an Object

Use QUICK.MAP to stop the approval process of a repository metadata object:

To stop approval on an object follow these steps.

1. Select the map for the object.
2. Select details.metaent.
3. Change the Approval Required selection to N.
4. Press PF3 to go back to the main Quick Map panel.
5. Select Process.

Select Save to save the change to the object. A message appears stating: Map has been saved to the control tables successfully.

1. Select OK.
2. Select Process.SAVE.

Data already pending approval will remain as pending when the approval process is stopped for an object type.

Setting Up Approvers

An approver is a group of users who can approve and reject changes to repository data. The Steward Workstation object is the Approver. Once the steward has been added to the repository, the steward must be related to a repository privilege. Individual users are then related to the privilege.

When a user goes to the Approval processing pages in Webstation Option, that user will only see objects that are related to any steward that is associated with the users repository privileges.

More Information:

See the chapters "Manage Repository Users" and "Working with Privileges" in this guide.

Approving an Object

When an update, insert, or delete is initialed inside WebStation Option, the application performs a task to check if the approval flag is on for the specified object.

If approval is required then the request is stored in the Approval Table. Multiple approvals can be pending.

When an approver selects the Approval Tab in WebStation Option, the application performs a task to mark all pending requests as valid or invalid.

An invalid request can occur when conflicts exists between pending requests as follows.

- Multiple updates for the same data. An approval of one update will cause all other requests for that row to be marked as invalid.
- An insert request for a row that already exist is invalid.
- A delete request for a non existent row is invalid.
- Pending requests may become invalid due to batch processing or any other update method.

Requests Validation Process

The validation process validates each Waiting Approval Request against the repository with respect to insert, delete, and update rules described above and the requests that failed validation will be marked as invalid request(IR) in the Approval Table. When the approver selects the Approval Page screen, each invalid request appears on the screen with the addition of the icon - Invalid Request. IR's can't be approved. Once a request is approved, the row is removed from the Approval Table. IR's will stay in the Approval Table until manually removed by the approver.

Approval Table

The following describes the Approval table.

Column Name	Data Type	Description
APPROVAL_IDENTITY_COL	INTEGER	Unique Id using by WSO
ENT_ROWID	ROWID	Unique Id
ENT_ID	INTEGER	Entity Id

ENT_TYPE	SMALLINT	Entity Type
NAME	VARCHAR(245)	Name
STATUS	CHAR(8)	Status
VERSION	SMALLINT	Version
REQUEST_TIMESTAMP	TIMESTAMP	Timestamp of this request
XREF_SEL_TIMESTAMP	TIMESTAMP	Timestamp from XREF table
ACTION	CHAR(1)	I – Insert, U – update, D – delete
OBJ_TYPE	CHAR(1)	E – entity, R – relationship, A - association
REQUEST_USER	CHAR(8)	User requesting approval
T_NAME	VARCHAR(245)	Target Name (relationship)
T_STATUS	CHAR(8)	Target Status(relationship)
T_VERSION	SMALLINT	Target Version(relationship)

Column Name	**Data Type**	**Description**
WORK_STATION_ID	VARCHAR(245)	Steward Name
LOGICAL_DELETE	CHAR(1)	` ` – waiting approval `L' – invalid request
DESCRIPTION	VARCHR(512)	Description
REQUEST_DOC	CLOB(1073K)	All of fields from object in XML format
CREATE_BY	CHAR(8)	
MOD_BY	CHAR(8)	
CREATE_DATE	DATE	
CREATE_TIME	TIME	
MOD_DATE	DATE	
MOD_TIME	TIME	

Appendix A: Utilities

This appendix contains information about the following CA Repository utilities:

Utility	Description
PRMORPH	Reports on and deletes incomplete repository data
PRMSTAT	Reports on, and optionally deletes, all repository data within a specified status

This section contains the following topics:

PRMORPH Utility (see page 325)
PRMSTAT Utility (see page 328)
RRSPB Utility (see page 330)

PRMORPH Utility

Use the PRMORPH utility to report on and delete incomplete data from the repository.

WARNING! Data corruption can occur when the repository is updated outside of the standard APIs.

PRMORPH either reports on errors and/or deletes corrupted data, and processes the following data:

Data Processed by PRMORPH	Description
Orphaned XREF rows	An XREF row is orphaned when there is a relationship or association row in the DBX_XREF table and either the source or target object is not defined in the DBX_XREF table.
Orphaned Entities	An instance of an entity is orphaned when there is a row defined as an entity in one of the metadata tables, and no row exists for the instance in the DBX_XREF table. An entity is also orphaned if a row exists in the DBX_XREF table for an entity and there is no corresponding row in the entity metadata table.
Orphaned	An instance of a relationship is orphaned when there is

Data Processed by PRMORPH	Description
Relationships	a row defined as a relationship in one of the metadata tables, and no row exists for the instance in the DBX_XREF table. A relationship is also orphaned if a row exists in the DBX_XREF table for a relationship and there is no corresponding row in the relationship metadata table.
Orphaned Workstation XREF rows	A workstation XREF row is orphaned when there is a row in the DBX_WKSN_XREF table and no corresponding row in the DBX_XREF table. A workstation XREF row is orphaned when there is a row in the DBX_WKSN_XREF table but no corresponding row in the DBX_WORKSTATION_D table.
Orphaned Text	Text is orphaned when there are rows in the text tables, but no corresponding row in the DBX_XREF table.
All of the above	

Run PRMORPH

PRMORPH can run in either of the following modes:

- Report mode—When running in report mode, the utility will report any data corruption found.
- Commit mode—When running in commit mode, PRMORPH will report and delete any data corruption found. Note that a delete may cause additional corruption to become apparent, for which no report entry was produced.

Example of PRMORPH Results

In a sample PRMORPH run, there may be both text and an XREF row for an entity for which there is no corresponding row in the entity metadata table. If you ran PRMORPH for orphaned text, the row would not show up as orphaned. It will show up after running with the orphan XREF rows option. In the same sense, deleting a row in the XREF table may cause other orphaned rows to show up within the XREF table.

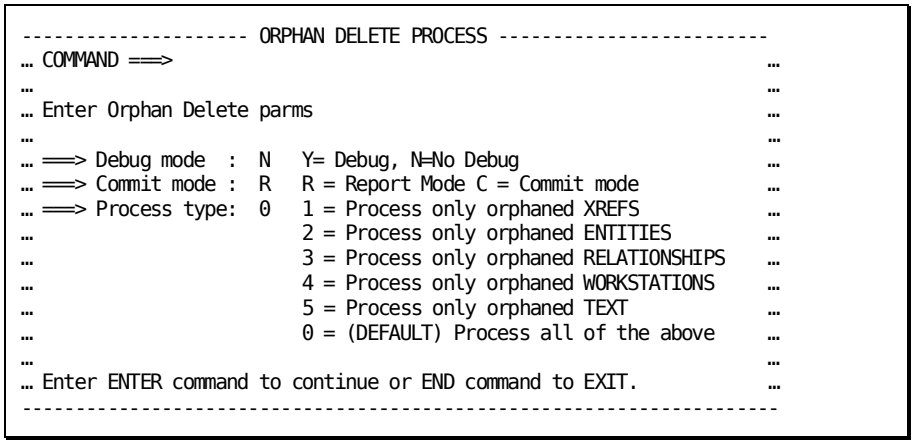
To ensure that there is no data corruption you should continue running PRMORPH until no errors are reported.

Access PRMORPH

It is not necessary to be in the repository to execute this command.

To access PRMORPH

1. Execute TSO ARZUTIL from the command line.
2. From the Utility panel, select Orphan Delete.



3. Enter values into the fields as follows:

Parameter	Description
Debug mode	Y-Yes run in Debug mode N- No Do not run in Debug mode
Commit mode	R-Run in Report mode C-Run in Commit mode
Process type	1-Process orphaned XREF rows 2-Process orphaned Entities 3-Process orphaned Relationships 4-Process orphaned Workstation XREF rows 5-Process orphaned Text 0-Run with all of the above options

4. Submit the generated JCL.

PRMSTAT Utility

Use the PRMSTAT utility to report on, and optionally delete, all data within a specified status.

The PRMSTAT utility:

- Does not delete the status from the control tables
- Does not delete any data from the workstation table (DBX_WORKSTATION_D)
- Deletes only from objects that are associated with a dialog or Product map
- Calls the IO module to retrieve and delete objects from the repository, therefore all referential integrity is preserved

When PRMSTAT is run in Report mode, the report does not list all relationships that are deleted. PRMSTAT does not report on objects that are deleted by the IO module to maintain referential integrity. PRMSTAT processes objects that are of the type entity first. When a repository instance is deleted, all relationships and associations to that instance are automatically deleted by the IO module.

- Example: The Object Table (Entity) is related to Object Element (Entity) through Object Column (Relationship).
- When an instance of Table is deleted, all columns for that table are automatically deleted by the IO module.

Run PRMSTAT

PRMSTAT can run in either of the following modes:

- Report mode

  When running in report mode, the utility reports on objects to be deleted.

- Commit mode

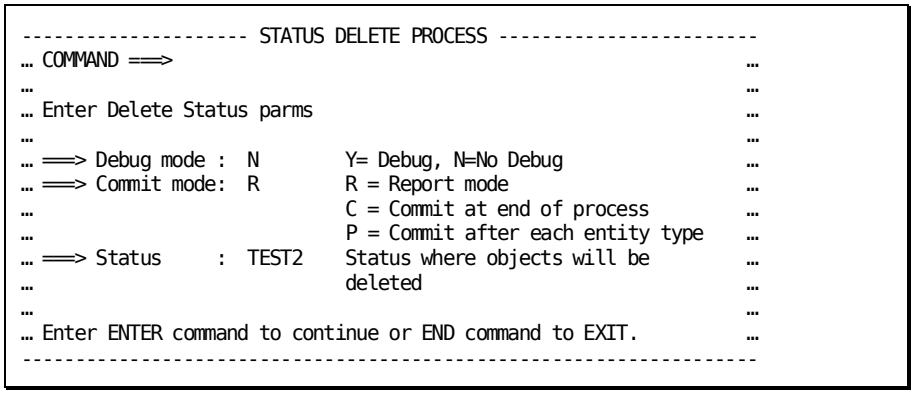
  When running in commit mode, PRMSTAT reports and deletes data in the specified status. Commits can be performed after each object type is processed or once at the end of the job.

Access PRMSTAT

It is not necessary to be in the repository to execute this command.

To access PRMSTAT

1. Execute TSO ARZUTIL from the command line.
2. From the Utility panel, select Delete objects by status.



3. Enter values into the fields as follows:

Parameter	Description
Debug mode	Y-Yes run in Debug mode N-No Do not run in Debug mode
Commit mode	R-Run in Report mode C-Commit once at the end of the entire process P-Commit after processing each entity type
Status	The name of the status that contains the objects to be deleted.

4. Submit the generated JCL.

RRSPB Utility

The Reuse Rule Stored Procedure Builder performs the following actions to build RRSPs:

1. Reads and validates the PCR file and generates SQL procedure text.
2. Calls the DB2 Stored Procedure Processor DSNTPSMP for building Stored Procedure.

   If the SQLCODE is not equal to 0, then

   For SQLCODE = 466,

   Prints result set from DSNTPSMP and stops the processing.

   For any other SQLCODE,

   Prints the error information using DB2 Error Reporting Routine (DSNTIAR) and skips all future calls to DSNTPSMP

DB2 Setting

Important: Consult with DB2 Administrator before running DSNTPSMP and WLM.

DB2 SQL procedure processor (DSNTPSMP) is required for generating and building RRSP.

Following sample shows 3 DDs use in DSNTPSMP:

```
//SQLDBRM DD DISP=SHR, <== DBRM Library
// DSN=db2sub.DBRMLIB.DATA
//SQLCSRC DD DISP=SHR, <== Generated C Source
// DSN=db2sub.SRCLIB.DATA
//SQLLMOD DD DISP=SHR, <== Application Loadlib
// DSN=db2sub.RUNLIB.LOAD
```

All of these 3 datasets could be customize but SQLLMOD dataset has to be included into WLM procedure as steplib concatenation.

Note: When running the RRSPB Utility, running out of space errors such as E37, B37 for the above mentioned datasets may occur.

When maintaining more than one repository for the same DB2 subsystem do the following changes to the settings in the DB2 Stored Procedure Processor (DSNTPSMP):

1. Create a new instance of DSNTPSMP with SCHEMA = &REPOWNER for each repository in same DB2 subsystem by copying the DSNTPSMP from IBM default member.
2. Make the following changes to the DDs:

```
//SQLDBRM  DD  DISP=SHR,                        <== DBRM Library
//             DSN=D91A.DBRMLIB.&REPOWNER.DATA
//SQLCSRC  DD  DISP=SHR,                        <== Generated C Source
//             DSN=D91A.SRCLIB.&REPOWNER.DATA
//SQLLMOD  DD  DISP=SHR,                        <== Application Loadlib
//             DSN=D91A.RUNLIB.&REPOWNER.LOAD

//**** DSNTPSMP Configuration File - CFGTPSMP (optional)
//*             A site provided sequential dataset or member, used to
//*             define customized operation of DSNTPSMP in this APPLENV
//CFGTPSMP DD  DISP=SHR,DSN=datasetname(CFGTPSMP)
```

3. Create a set of new datasets for DDs:

   SQLDBRM

   SQLCSRC

   SQLLMOD

4. Obtain DCB information of that datasets from IBM DSNTPSMP JCL

5. Create dataset for DD CFGTPSMP

   With RECFM=F and Lrecl=80

   datasetname(CFGTPSMP) should be consists of one line coding from first position:

   EXTERNAL_UNIQUENESS = LOADLIB

6. Execute following SQL statement:

```
CREATE PROCEDURE &REPOWNER..DSNTPSMP
  ( IN  FUNCTION_REQUEST   VARCHAR(20)
  , IN  SQL_ROUTINE_NAME   VARCHAR(261)
  , IN  SQL_ROUTINE_SOURCE CLOB(2M)
  , IN  BIND_OPTIONS       VARCHAR(1024)
  , IN  COMPILE_OPTIONS    VARCHAR(255)
  , IN  PRECOMPILE_OPTIONS VARCHAR(255)
  , IN  PRELINK_OPTIONS    VARCHAR(255)
  , IN  LINK_OPTIONS       VARCHAR(255)
  , IN  ALTER_STATEMENT    VARCHAR(32672)
    , IN  SOURCE_DATASETNAME VARCHAR(80)
    , IN  BUILDOWNER         VARCHAR(130)
    , IN  BUILDUTILITY       VARCHAR(261)
    , OUT RETURN_VALUE       VARCHAR(255) )
   PARAMETER CCSID EBCDIC
   RESULT SET 1
   EXTERNAL NAME DSNTPSMP
   LANGUAGE REXX
PARAMETER STYLE GENERAL
MODIFIES SQL DATA
    COLLID DSNREXCS
      WLM ENVIRONMENT  wlm launching DSNTPSMP
      ASUTIME NO LIMIT
      STAY RESIDENT NO
      PROGRAM TYPE MAIN
      SECURITY USER
      RUN OPTIONS 'MSGFILE(SYSTSPRT,,,,NOENQ)'
      COMMIT ON RETURN NO;
      GRANT EXECUTE ON PROCEDURE &REPOWNER.DSNTPSMP TO PUBLIC;
 COMMIT;
```

   Note: You can find value of &REPOWNER from Installation Menu on Define DB2 Install Options as:
   Repository Owner ===>

More Information:

See DB2 for z/OS® SQL procedure processor (DSNTPSMP) refers to IBM DB2 UDB manual '*Application Programming and SQL Guide.*'

Run RRSPB

RRSPB can run in either of the following modes:

- Generate mode

  When running in generate mode, the stored procedures will be generated as SQL language source code in output dataset.

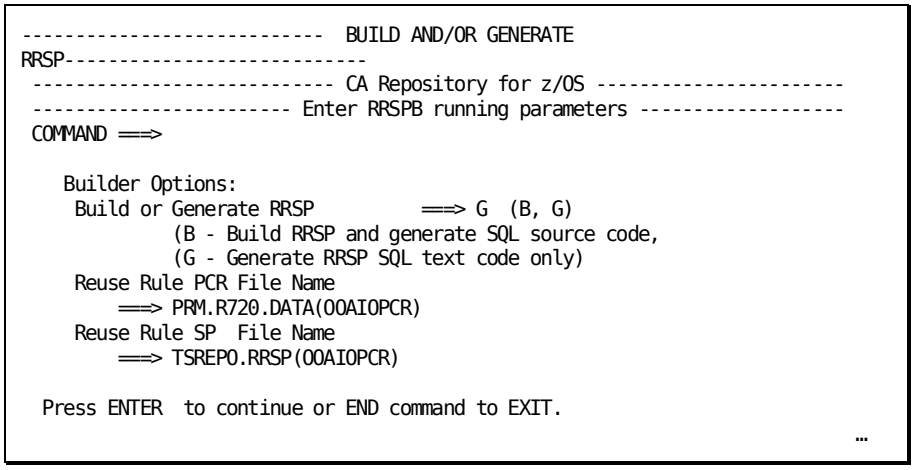
- Build code

  When running in build code mode, the stored procedures will be run in generate mode and then these build stored procedure will be saved in the DB2 runlib library.

Access RRSPB

To access RRSPB

1. Execute TSO ARZUTIL from the command line.
2. From the Utility panel, select Build and/or generate RRSP.



3. Reuse Stored Procedures will be created with following default parameters:

   SCHEMA as &REPOWNER

   PACKAGE as &PRMPCKG

   WLM as &WLMNAME

   Note: you can find value of &REPOWNER, &PRMPCKG and &WLMNAME from Installation Menu on Define DB2 Install Options as:

   Repository Owner ===>

   WLM environment Name ===>

   Repository Package Collection ===>

4. Enter values into the fields as follows:

Parameter	Description
Build or Generate RRSP	B - Build RRSP and Generate SQL source code. G - Generate RRSP SQL source code only.
Reuse Rule PCR File Name	The name of PCR file with Reuse rules. Dataset should be as ps, pds or pdse and lrecl=235.
Reuse Rule SP File Name	The Name of dataset where will be generate source code of stored procedures. Dataset should be preallocate as pds or pdse and lrecl=80.

5. Submit the generated JCL.

Index

A

access to repository, granting to DB2 users • 44
ADD option, migration extension • 277
adding
  a code table • 119
  a code table to the control tables • 124
  a migration path to the control tables • 81
  a new status to the repository • 77
  a user to the control tables • 51
  commands to control tables • 92
  commands to the repository • 88
  input fields to maps • 99, 100
  keywords • 146
  message text to a message • 138
  migration paths • 78
  privileges • 57
  privileges to control tables • 63
  screen literals to maps • 99
  variances to a code table • 122
APIs
  CRTXT80
example • 186
input • 181
input-output • 184
output • 184
  DBXIOD
example • 189
input • 193
output • 193
  DBXIOI
example • 199
parameters • 195
  DBXIOM
example • 202
input • 201
output • 201
  DBXIOR
example • 204
input • 203
output • 203
  DBXIOU
example • 221
input • 217
output • 219
APIs, CA Repository z/OS
  CGETVER • 176
  CRTXT80 • 181
  DBXIOD • 193
  DBXIOI • 194
  DBXIOM • 200
  DBXIOR • 203
  DBXIOS • 205
  DBXIOU • 216
  DBXNSVI • 222
  DBXSEC • 224
  summary table • 175
application life cycle • 73
associations between entities, using maps to create • 107
attribute concatenation • 247
attribute statements, path • 293

B

back up
  of components • 263
backup
  and recovery procedures • 263
batch load
  Action parameter • 236
  card • 233
  Data Source parameter • 236
  deleting metadata • 255
  destination parameters • 243
  Error parameter • 238
  error SQL -811 • 249
  example
DELETE • 255
INSERT, loading metadata example • 250
INSERT, loading metadata into associations • 251
REUSE • 253
SELECT • 254
UPDATE • 252
  exporting metadata • 254
  FIELD declaration parameters • 243
  fields, format-block • 244
  FROM parameter • 237
  keyword FIELDS • 244
  loading metadata • 250
  loading metadata into associations • 251
  LOG keyword • 242

multiple load cards • 249
required parameters • 236
reusing metadata • 253
sample of syntax • 248
SAVEAUDIT parameter • 241
SEARCHSTATUS2 parameter • 241
SKIPBLANK parameter • 241
special keywords • 246
STATUS parameter • 240
syntax, illustration of • 235
updating metadata • 252
VALIDATE parameter • 240
version parameter • 239
WORKSTATION parameter • 242
workstation processing • 256
Batch Load Facility
overview • 233
BUILD Extend command, using • 35

C

CA Repository
creating paths • 283
tuning the database • 264
CA Repository for z/OS
access and support • 16
implementation and customization • 17
maintenence • 24
required knowledge • 16
targeting usage areas • 17
CA Repository z/OS or APIs
CGETVER • 176
CRTXT80 • 181
DBXIOD • 193
DBXIOI • 194
DBXIOM • 200
DBXIOR • 203
DBXIOS • 205
DBXIOU • 216
DBXNSVI • 222
DBXSEC • 224
summary table • 175
calculating map offsets • 114
calling pattern for I/O module • 152
CASE tools • 17
catalog synchronization
during relational translator process • 305
CGETVER API
example • 179
input • 176
output • 177
checklist of commands • 67
code table
adding • 119
adding to the control tables • 124
adding variances to • 122
defining • 119
extracting data from • 125
illustration of • 118
modifying • 125
providing help for • 123
removing • 126
understanding the model • 118
CODE TBL Edit screen • 124
COMMAND EXIT PROCEDURE input fields • 313
COMMAND input fields • 89
Command model
concepts, illustration • 87
commands
adding • 88
adding to control tables • 92
adding to the repository • 89
BUILD, code table • 124
checklist of • 67
defining • 89
IMPORT, code table • 125
limiting for specific entity types • 91
modifying • 93
providing help with • 91
REMOVE, code table • 126
removing • 94
control tables
adding commands • 92
adding messages to • 140
adding migration paths • 81
adding privileges • 63
adding users • 51
affected by the I/O module • 167
attribute data in Relational Translator • 306
Entity, in Relational Translator • 305
extracting data from • 52
importing data • 52
importing data from to storage tables • 82
layout information for Relational Translator • 310
removing users from • 65
use of for parameter storage • 18
COPY option, migration extension • 278
CREATE Extend command, using • 41

creating
DB2 data structures req'd to store repository objects • 41
multiple names for a single object, creating • 23
on-screen links • 107
paths • 287
cross-references, Relational Translator • 301
customizing
characters used for input fields, QuickMap • 101
messages • 140
the repository • 17
tools in the Extend model • 18

D

dashes, spaces, and underscores, using • 24
DBXIOS API
example • 209
input • 206
output • 208
DBXNSVI API
example • 224
input • 222
output • 223
DBXNSVI stored procedure • 222
DBXSEC API
example • 227
parameters • 225
deactivating a repository component by removing • 40
deactivating keywords • 149
default
privilege feature, activating • 66
status, batch load • 240
version parameter • 239
defining
a command instance • 89
a variance • 121
default privileges • 65
entity type attribute details • 102
privileges • 57
deleting
metadata, batch load • 254, 255
paths • 284
dialog, definition of • 33
domain
creating a domain dialog that points to domain entity types • 130
keys and values, concepts • 127
keys, definition • 127
sets definition • 127
sets, using • 127
tables • 129
values, definition • 127
domain dialog, creating • 130
domain tables
concepts • 129
creating • 130

E

editing
keywords • 147
element re-usability, Relational Translator • 302
Enable the History Feature • 265
ENT MIG input fields • 79, 80
Entity control table • 305
entity name generation • 302
entity type attributes, defining • 102
entity versions • 23
executing VIEWMSG • 41
exporting
metadata, batch load • 254
Extend commands
BUILD, using • 35
CREATE • 41
GRANT • 44
IMPORT • 38
NEXTID • 43
object types • 34
overview • 34
RECREATE • 42
REMOVE • 40
REVOKE • 45
VIEWMSG • 41
Extend Facility • 19
extend keyword reference tables • 149
Extend model • 18
component descriptions • 25
components • 25
dialogs, concepts • 33
managing repository users • 47
object types for defining illustration • 55
overview • 25
sub dialogs (models) • 33
sub-dialogs
COMMAND • 33
DIALOG • 33

ENTITY • 33
HELPMSG • 33
MAP • 33
PRIV • 33
STATUS • 33
USERS • 33
    using to add or modify commands • 87
    using to customize the repository • 25
extensibility
    Extend Facility • 19
    guidelines • 19
extensions
    types of • 19
extracting
    altered privilege data • 64
    data from a code table • 125
    user data from control tables • 52

F

functions
    Get_streamtext_32K • 228

G

generating names • 302
Get_streamtext_32K function
    example • 230
    input • 229
    output • 229
    overview • 228
GRANT Extend command, using • 44
granting DB2 access to users • 44
granting privileges to a user • 49

H

help
    messages, adding a • 135
    providing for a code table • 123
    providing information and messages about a command • 91
help messages
    add or customize, Extend model • 41
    migration paths • 80
history core tables • 266
history feature, enabling • 265
history objectremove • 116
history tables
    overview • 265
history trail processing, special considerations • 266

I

I/O module
    calling pattern • 152
    calling pattern for • 152
    control tables affected by • 167
    decimal fields • 163
    DELETE statements • 164
    INSERT statements • 164
    map selection • 162
    maps • 161
    overview • 151
    processing messages • 162
    REPLACE statements • 165
    UPDATE statements • 164
    using to manipulate data • 151
I/O module variables
    BLOCK • 156
    elements of • 154
    MAPAREA • 158
    MSG_ID • 160
    MSG_QUAL • 161
    ORDER_BY • 158
    REASON • 159
    RETURN_CODE • 160
    SQLAREA • 159
    WHERE DATA • 157
illustrations
    batch load fields, format block • 244
    batch load syntax • 235
    code table • 118
    Command model concepts • 87
    Extend model object types for definingprivileges • 55
    keywords and maps • 145
    Relational Translator • 299
    Relational Translator, tie-back cross referencing • 301
    status model • 75
    statuses with level numbers • 73
    Storage Table and Domain Table Combination • 129
    User-Privilege Model • 47
implementation of the repository • 17
IMPORT Extend command, using • 38
importing
    data from control tables to recreate keyword data • 147
    data from control tables to storage tables • 82

from object types • 38
message data from the control tables • 141
user data from control tables to storage tables • 52
input fields
adding to maps • 99
inserting
a user instance into the repository • 48
ISPF variables available in user exits • 316

K

keywords
adding • 145, 146
deactivating • 149
editing • 147
for the Extend facility • 149
illustration of • 145
importing data from control tables to storage tables • 147
modifying • 147
removing • 149

L

level number • 73
level numbers, understanding status • 73
limiting
commands for specific entity types • 91
migration paths • 79
privileges to entity types • 61
linking
a map to an entity type • 111
maps to a dialog • 113
privileges to commands • 58
privileges to dialogs • 60
list of control table definitions • 38
loading
metadata into associations, batch load • 251
metadata, batch INSERT • 250
logical components, supporting via metadata • 22

M

maintaining metadata in the repository • 22
maintenance, on repository • 24
MAP and ENTITY IDs, determining availability • 43
MAPAREA variable • 158
maps
adding input fields to • 100
adding screen literals • 99
calculating map offsets • 114
customizing • 95
linking to a dialog • 113
linking to an entity type • 111
saving • 115
specifying storage tables for • 110
using Quick Map • 95
using to create relationships between entities • 107
MESSAGE input fields, entering values in • 136
Message Model
components • 135
overview • 135
messages
adding to control tables • 140
customizing • 140
customizing the LOGON message • 143
importing data from the control tables • 141
inserting a MESSAGE instance • 136
model of • 135
removing • 144
supplemental, adding • 139
types
in CA Repository • 133
types, table of • 133
using the ISPF Edit screen to add rows • 138
variables and format commands • 138
metadata
maintaining in repository • 22
methods of populating the repository • 21
security controls • 20
security guidelines • 21
metadata history file selection
general notes • 269
using EXTEND.MODEL • 271
using QUICK.MAP • 270
metadata security
implementation options • 20
MIG PATH input fields • 78
migrating user updates • 277
ADD option • 277
COPY option • 278
REPLACE option • 278
migration
specifying entity comparison • 102
migration extensions
migrating user updates • 277
migration paths • 73

adding • 78
adding to control tables • 81
help messages • 80
limiting • 79
modifying • 82
removing statuses • 84
with statuses • 75
migration user extensions • 279
modifying
a message • 140
commands • 93
keywords • 147
privileges • 63
statuses and migration paths • 82
users • 51
MSG_ID variable • 160
multiple names for a single object, creating • 23
multiple subsystem support • 16

N

name generation standards • 24
Name, Status, Version
DBXNSVI stored procedure
DBXNSVI stored procedure • 222
naming
entity versions • 23
multiple names • 23
standards • 23
using prefixes • 23
NEXTID Extend command, using • 43

O

on-screen links, example of creating • 109
opening
Quick Map • 96
ORDER_BY variable • 158
orphaned or incomplete data, processing • 331

P

parameters
batch load syntax • 236
Path Delete Facility • 284
path facilities
considerations • 297
Path Delete • 284
Path Report • 283
Path Workstation Add • 285
specifying parameters • 295
Path Reports facility • 283
Path Workstation Add facility • 285
paths
attribute statements • 293
creating • 287
definition of in CA Repository • 283
definitions
adding or modifying • 287
creating statements • 288
delete statement • 292
example of statements • 290
deleting • 284
executing report facility • 284
loading • 287
overview of how processed • 287
physical components, supporting • 22
populating
the control tables using Extend model object defs • 35
the repository with metadata • 21
prefixes, defining • 23
PRIV Edit screen • 63
PRIV STS associations • 62, 63, 82
privileges
adding • 57
adding to control tables • 63
associating users in the repository • 49
concepts • 55
default user • 65
defaults, activating • 66
defining • 57
limiting to entity types • 61
linking to commands • 58
linking to dialogs • 60
model, understanding • 55
modifying • 63
purpose of entity and assoc. types • 55
removing • 65
PRMMODL utility
overview • 273
unload control data • 275
PRMORPH utility
overview • 331
running • 332
product-level history file activation
after completing installation • 269
overview • 267
when installing for the first time • 268
when upgrading • 269
PRSTAT utility

overview • 334
running • 334

Q

Quick Map
concepts • 95
opening • 96
using TSO edit commands • 96

R

REASON variable • 159
recovery procedures • 263
RECREATE Extend command, using • 42
recreating an existing DB2 storage table • 114
Relational Translator
Attribute control table • 306
attribute name generation • 302
concepts • 299
control tables, layout information for • 310
element re-usability • 302
entity and attribute control tables • 305
illustration • 299
logical models, selecting • 300
model requirements • 308
screen facility • 302
tie-back cross referencing • 301
tying the relational to the physical, illustration • 301
using logical dialog to create DBMS component • 309
viewing processing results • 305
workstation processing • 301
REMOVE Extend command, using • 40
remove history objects • 116
removing
a user from the control tables • 65
commands • 94
keywords • 149
messages • 144
privileges • 65
statuses • 84
users • 54
removing a code table • 126
REPLACE option, migration extension • 278
repository
maintaining metadata in • 22
populating using metadata • 21
repository entity types, overveiw • 95
requirements
reusing
metadata, batch load • 253
size • 16
REVOKE Extend command, using • 45
revoking user access to repository tables • 45
runtime parameters for SYSIN data card • 296

S

saving maps • 115
screen literals
SAMPLE ENTITY NAME • 99
STATUS • 99
VERSION • 99
security
and user access • 20
metadata • 20
security controls for metadata • 20
size requirements • 16
special formatting characters
Hilite literal • 101
Lolite literal • 101
using • 101
SQLAREA variable • 159
status
addinga new to the repository • 77
concepts • 73
DEVnn • 86
entities • 75
input fields • 77
level number restrictions • 74
level numbers • 73
level numbers, understanding • 73
modifying • 82
PROD • 85
removing • 84
TEST • 85
using to support projects • 85
validating • 78
status model
illustration of • 75
status model components
opening Edit screen for • 75
status, data
reporting and deleting • 334
Statuses
with level numbers, illustration • 73
stop auditing an object type • 271
storage table and domain table combination, illustration • 129

storage tables
DB2, recreate an existing • 114
specifying for maps • 110
stored procedure example • 172
stored procedure, DBXNSVI • 222
supplemental messages, adding • 139
support
for logical components • 22
for physical components • 22
multiple subsystems • 16
vendor tool • 17
support for internal and third-party tools • 17
SYSIN data card, parameters • 296
SYSTEM.EXTEND.BUILD command • 63, 82
SYSTEM.EXTEND.REVOKE command • 45

T

text
adding to messages • 138
Text Load facility • 259
tuning the database • 264
types of status that support projects • 85

U

updating metadata, batch load
example • 252
USER entity type • 52
user exits
adding • 313
concepts • 313
ISPF variables • 316
writing • 315
USERPRIV
entity type • 52
input fields • 49
User-Privilege Model, illustration of • 47
users
access and security • 20
access and TSO • 16
adding to control tables • 51
associating with repository privileges • 49
BUILD Command • 51
defining privileges • 65
inserting a user instance • 48
managing repository users using the Extend model • 47
modifying • 51
removing • 54
removing from the control tables • 54, 65
USERPRIV
associations • 51
utilities
PRMORPH • 331
PRSTAT • 334

V

variables
I/O module • 153
I/O module, elements of • 154
variances
adding to a code table • 122
defining • 121
version of an entity • 23
VIEWMSG Extend command, using • 41

W

where clause
I/O module • 163
WHERE_DATA variable • 157
work tables
in the Relational Translator • 305
workstation processing
batch load • 256
Relational Translator
Relational Translator • 301