ASG-MethodManager® Administration

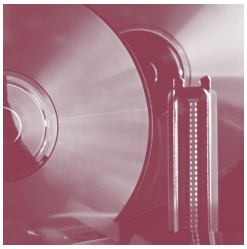
Version: 2.5

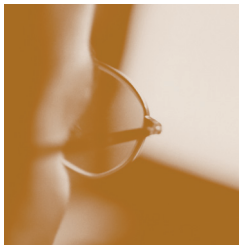
Publication Number: MMR2100-25-ADMIN

Publication Date: December 2000

The information contained herein is the confidential and proprietary information of Allen Systems Group, Inc. Unauthorized use of this information and disclosure to third parties is expressly prohibited. This technical publication may not be reproduced in whole or in part, by any means, without the express written consent of Allen Systems Group, Inc.

© 1998-2001 Allen Systems Group, Inc. All rights reserved.
All names and products contained herein are the trademarks or registered trademarks of their respective holders.





ASG Worldwide Headquarters Naples, Florida USA | asg.com
1333 Third Avenue South, Naples, Florida 34102 USA Tel: 941.435.2200 Fax: 941.263.3692 Toll Free: 1.800.932.5536

ASG Documentation/Product Enhancement Fax Form

Please FAX comments regarding ASG products and/or documentation to (941) 263-3692.

Company Name	Telephone Number	Site ID	Contact name

Product Name/Publication	Version #	Publication Date
Product:		
Publication:		
Tape VOLSER:		

Enhancement Request:

© 2001 Allen Systems Group, Inc.
All names and products are trademarks or registered trademarks of their respective holders.

ASG Support Numbers

ASG provides support throughout the world to resolve questions or problems regarding installation, operation, or use of our products. We provide all levels of support during normal business hours and emergency support during non-business hours. To expedite response time, please follow these procedures.

Please have this information ready:

- Product name, version number, and release number
- List of any fixes currently applied
- Any alphanumeric error codes or messages written precisely or displayed
- A description of the specific steps that immediately preceded the problem
- The severity code (ASG Support uses an escalated severity system to prioritize service to our clients. The severity codes and their meanings are listed below.)
- Verify whether you received an ASG Service Pack for this product. It may include information to help you resolve questions regarding installation of this ASG product. The Service Pack instructions are in a text file on the distribution media included with the Service Pack.

If You Receive a Voice Mail Message:

1. Follow the instructions to report a production-down or critical problem.
2. Leave a detailed message including your name and phone number. A Support representative will be paged and will return your call as soon as possible.
3. Please have the information described above ready for when you are contacted by the Support representative.

Severity Codes and Expected Support Response Times

Severity	Meaning	Expected Support Response Time
1	Production down, critical situation	Within 30 minutes
2	Major component of product disabled	Within 2 hours
3	Problem with the product, but customer has work-around solution	Within 4 hours
4	"How-to" questions and enhancement requests	Within 4 hours

ASG provides software products that run in a number of third-party vendor environments. Support for all non-ASG products is the responsibility of the respective vendor. In the event a vendor discontinues support for a hardware and/or software product, ASG cannot be held responsible for problems arising from the use of that unsupported version.

Business Hours Support

Your Location	Phone	Fax	E-mail
United States and Canada	800.354.3578 1.941.435.2201 **Secondary Numbers:** 800.227.7774 800.525.7775	941.263.2883	support@asg.com
Australia	61.2.9460.0411	61.2.9460.0280	support.au@asg.com
England	44.1727.736305	44.1727.812018	support.uk@asg.com
France	33.141.028590	33.141.028589	support.fr@asg.com
Germany	49.89.45716.300	49.89.45716.400	support.de@asg.com
Singapore	65.224.3080	65.224.8516	support.sg@asg.com
All other countries:	1.941.435.2201		support@asg.com

Non-Business Hours - Emergency Support

Your Location	Phone	Your Location	Phone
United States and Canada	800.354.3578 1.941.435.2201 **Secondary Numbers:** 800.227.7774 800.525.7775 **Fax:** 941.263.2883		
Asia	011.65.224.3080	**Japan/Telecom**	0041.800.9932.5536
Australia	0011.800.9932.5536	**New Zealand**	00.800.9932.5536
Denmark	00.800.9932.5536	**South Korea**	001.800.9932.5536
France	00.800.9932.5536	**Sweden/Telia**	009.800.9932.5536
Germany	00.800.9932.5536	**Switzerland**	00.800.9932.5536
Hong Kong	001.800.9932.5536	**Thailand**	001.800.9932.5536
Ireland	00.800.9932.5536	**United Kingdom**	00.800.9932.5536
Israel/Bezeq	014.800.9932.5536		
Japan/IDC	0061.800.9932.5536	**All other countries**	1.941.435.2201

ASG Web Site

Visit http://www.asg.com, ASG's World Wide Web site.

Submit all product and documentation suggestions to ASG's product management team at http://www.asg.com/products/suggestions.asp

If you do not have access to the web, FAX your suggestions to product management at (941) 263-3692. Please include your name, company, work phone, e-mail ID, and the name of the ASG product you are using. For documentation suggestions include the publication number located on the publication's front cover.

Contents

Preface xiii

About this Publication xiii

Publication Conventions xv

1 Introducing Administrative Roles 1

2 MethodManager Introduction 3

3 Repository Information Models 5

What is a Rim? 6

Modeling the Real World 7
Entity Types and Relationship Types 8
Entities and Relationships 10
Properties of Relationship Types 10

How a RIM Models the Real World 17
EA and ER Relationship Types 18
Naming Conventions 21

Defining a RIM 22
UDS Member Types 24
Panel-Interface Member Types 25
HDS-TABLE Member Type 26

Example of Implementing a RIM 26
Designing the RIM 27
Defining the RIM 29
The Enabled Environment 32

Checklist of Steps for Implementing a RIM 36

Managing META-DATA 38
Ensuring Integrity 39
Examples of Integrity Checking 40
Removing Members 41

Interrogating a Model 43

4 Defining the Panel Interface 45

Defining Panels of Different Types 48
Menus 48
Input Panels 51
List Panels 55
Output Panels 59

Tailoring Panels of the Update Cycle 61
Using Assisted Update on Views 63
Example of Using a View 64
MMRVIEW Command 65
AUPDATE Command 67

Defining Help 68
Defining Extended Help in INFOBANK-PANEL or FMT-SCREEN Members 69
Defining Extended Help in Any Member Type Other Than INFOBANK-PANEL or FMT-SCREEN 72
Defining Contextual Help in ITEM Members 75
Defining Contextual Help in Any Member Type of the Repository 80

Enabling a Defined Panel 87
CX Command Syntax 87

5 Enabling the Environment 89

Enabling HDS Tables 92

Complete Generation 98

Partial Generation 99

Analyzing Generated Executives 99

The UX COMMAND 101
UX Command Syntax 101

6 Customizing the Environment 103

Global Variables Defined in ITEM Members 106

Global Variables Defined in SEXEC Members 107
Customizing Functional Areas Using Global Variables 107
Standard String Delimiter: MDG_STADEL 109
Secondary String Delimiter: MDG_SECDEL 109
Translation of Alphabetic Characters: MDG_UPDLOW 110
Translation of Internal Keywords: MDG_MIXED1, MDG_MIXED2 110
Clause Separator: MDG_ATTSEP 111
Line Erase Character(s): MDG_DELSTR 112

Blank String Character(s): MDG_BLASTR 112
Keyword Indicator: MDG_UPDHEAD 112
Offset for Member Type Alias: MDG_SYMOFF 113
Specifying Prompt Formats. 113
Standard Prompt: MDG_SKSTR2 114
Time Prompt: MDG_SKSTR3 114
Date Prompt: MDG_SKSTR4. 114
Alias Prompt: MDG_SKSTR5 115
Compulsory Input Prompt: MDG_SKSTR6. 115
Selection Prompt: MDG_SKSTR7 115
Line Protection Character: MDG_LINE_PROTECTION_CHAR. 116
Hexadecimal Code of Line Protection Character: MDG_LINE_PROTECTION_CODE . 117
Formatting Process Indicator: MDG_AUPD_AMEND and Formatting Process Bypass Array: MDG_AUPD_AMEND_EXCLUDE (N) 117

Customizing the Command Interface 118
Autoskip Feature: MDG_MMR_SET_AUTOSKIP 118
Buffer Limit: MDG_MMR_SET_BUFFER_LIMIT 118
Retention of Lookaside Buffers: MDG_MMR_SET_LOOKASIDE_RETENTION. . 119
Retention of Line Commands: MDG_MMR_SET_LINEAR_RETENTION 119
Condition for Update Output: MDG_MMR_SET_UPDATE_OUTPUT. 120
Position of Line Command Area: MDG_MMR_SET_LINE_COMMAND 120
Position of Command Area: MDG_MMR_SET_COMMAND_LINE 121
Output Line Limit: MDG_MMR_SET_OUTPUT_LINES 121
Panel Limits: MDG_MMR_SET_PANEL_LIMITS 122
Position of the Command Area for a Single Panel Type: MDG_MMR_CX_CMD_LINE(N) 123
Panel Type for which a Command Area is to be Generated: MDG_MMR_CX_CMD_TYPE(N) 124
Control Whether Panel Invokes the Panel Display Exit (EC0995): MDG_GEN_PANEL_EXIT 125
Character that Marks an Input Field on a Panel: MDG_TABLE_FIELD_CHAR 125
Character that Marks the Command Area on a Panel: MDG_COMMAND_LINE_CHAR 125
Character that Marks the Line Command Area on a Panel: MDG_LINE_COMMAND_CHAR 126
Enable/Disable Automatic Logoff from Manager Software Products: MDG_LOGOFF . . 126
Number of Columns a Member Name is to be Indented in a Relationship Display: MDG_STINC 126
Maximum Depth for the USA and REFA Line Commands: MDG_STMAX 127
Separator Between Member Name and Level Number in a Relationship Display: MDG_STSEP 127
Maximum Number of Columns of a Matrix Displayed Online: MDG_MATRIX_SIZE_ONLINE 127

Maximum Number of Columns of a Matrix Processed in Batch: MDG_MATRIX_SIZE_BATCH 128
User-Definable Areas on Panel: MDG_USER_AREA_1 and MDG_USER_AREA_2 128

Customizing the Documentation Functions 129
Enable or Disable Copy Function of DCUPD Command: MDG_DOKINC 130
Clause Defining the Body of a Document: MPR_EA60_DBODY 131
Clause Defining the Heading of a Document: MPR_EA60_HEADING 131
Enable or Disable Automatic Composition of Complex Documentation from Several Levels of Sub-documents via the ??INCLUDE Command: MPR_EA60_DECOMPOSE 132
Enable or Disable Automatic Numbering of Headings: MPR_EA60_INDEX 132

Customizing Naming Conventions of Members. 133
Wildcard for Maximum Length of a Member Name: MDG_NAMEOL 133
Wildcard for Exact Length of a Member Name: MDG_NAMEON. 133
Wildcard for Minimum Length of a Member Name: MDG_NAMSOL 134
Wildcard for a Mandatory Alphanumeric or Special Value: MDG_NAMJOK 134
Wildcard for a Numeric Value: MDG_NAMNUM 134
Wildcard for the Variable Part of a Member Name: MDG_NAMVAR. 135
Wildcard for any Number of Optional Alpanumeric Values: MDG_NAMOPT 135
Enable/Disable Assisted Update for Existing Members with Invalid Naming Convention: MDG_NAM_OLD 136
Specify Existing Member Type(s) with Invalid Naming Convention for which the Assisted Update is Enabled or Disabled: MDG_NAM_OLD_MEM(N) 136
Specify Standard Names and Abbreviations for Repository Members: MDG_NAM_STD_NAME(N) and MDG_NAM_STD_ABBREV(N) 137
Enforce Standard Member Names in Assisted Update: MDG_NAM_ENFORCE . . . 138
Enforce Naming Conventions for Dummy Members: MDG_NAM_NEW 139
Enable/Disable Naming Conventions throughout the Repository: MDG_NAMTST . 139

Customizing the Retain Options 140

Customizing the Workbench Design Area 141
Enable or Disable ITEM Member Check: MDG_WBDA_ITEM_CHECK. 141
Enable or Disable Replacement of Substring in Naming Convention of ITEM Members: MDG_WBDA_ITEM_REPLACE 141
Indicator of Substring to be Replaced in Naming Convention of ITEM Members: MDG_WBDA_SWITCH_PRSU_IT 142
Existing Prefix of ITEM Member Name that is to be Replaced: MDG_WBDA_ITEM_PREF_OLD 142
New Prefix that Replaces Existing Prefix of ITEM Member Name: MDG_WBDA_ITEM_PREF_NEW. 143
Existing Suffix of ITEM Member Name that is to be Replaced: MDG_WBDA_ITEM_SUFF_OLD 143
New Suffix that Replaces Existing Suffix of ITEM Member Name: MDG_WBDA_ITEM_SUFF_NEW. 144

Character that Initiates the Generation of a Default Identifier Name for the Data Element of an Entity: MDG_WBDA_RHSPRE 144
Name of User-defined Member Type Defining an Object of a DB2 or SQL/DS Database System: MDG_WBDA_TABLE_TYPE(N) 145
Indicator of Naming Conventions for Members Generated from Objects of a DB2 or SQL/DS Database System in the WBDA: MDG_WBDA_TABLE_PRSU(N) 147
Name of User-defined Executive Routine: MDG_WBDA_NAMING_EXIT 148

Activating User Exits for Toolset Services 149

Customizing Return From Buffers 150

Customizing Life Cycle Services (LCS) 151
Customizing Member Types Relevant for Life Cycle Services 151
Customizing Clauses of Member Types Relevant for LifeCycle SERVICES 154
Customizing Relationships Between Member Types Relevant for LifeCycle Services 158
Customizing Panels Used Under Life Cycle Services 161
Customizing Project Management Functions of Life Cycle Services 163
Customizing Clauses Defining the Duration of a Task or a Project 166
Activating User Exits for Life Cycle Services 168

Customizing - Miscellaneous 169

7 User Exits 171

Global Exit Routines 172
Tailoring the Naming Convention Process 174
Tailoring the Assisted Update Process 175
Tailoring the File Process 178
Tailoring the Display of Relationships Between Members 180
Tailoring Member Protection 181
Tailoring CX Processing 182
Tailoring Panel Processing 182
Tailoring the Panel Display 184

Life Cycle Services 185
Tailoring Panel Display within Life Cycle Model 185
Tailoring Project Definition 186
Tailoring Assignment of Life Cycle to Project 187
Tailoring Assignment of User to Project 187
Tailoring Project Selection 187
Tailoring Task Selection 188
Tailoring VX/VXA Processing 188
Tailoring the Naming Convention Process 189
Tailoring the Assisted Update Process 191
Tailoring the File Process 192

Dynamic Exit Routines 193
Tailoring the Return to the Panel Interface 193

8 Macros 195

Macro Descriptions 198
:BROWSE 198
:FMTSCREEN 199
LPARM 200
NAMKO 201
NAMKOT 206
OUTE 207
RETAIN 208
VCHNG 209
VSEARCH 210
XFILE 211

9 Member Types 215

ATTRIBUTE-GROUP 218
Specifying the ATTRIBUTE-TYPE members Contained in the Group 219
ATTRIBUTE-GROUP Syntax 220

ATTRIBUTE-TYPE 220
Defining the Name of a Clause or Identified Keyword 221
Defining the Name of an Unidentified Keyword 222
Defining the Type of Value Permitted 222
Defining Specific Permitted Values 225
Defining the Permitted Number of Values 226
Defining Minimum and Maximum Permitted Values 227
Defining Installation Independent Date and Time Values 227
Defining the Length of a Value 229
Defining the Number of Lines of Text that can be Entered in a Clause 229
Indexing User-defined Attributes by Presence or Value 229
Renaming UDR and UDRS Clauses and Displaying Clauses with Identifiers Containing More than One Keyword 231
Defining a Line of Help in an Assisted Update Buffer 233
Defining an Assisted Update Buffer Input Prompt 233
Defining a Complex Assisted Update Buffer Input Prompt 235
Defining How Clauses and Keywords are Formatted by Assisted Update 236
Taking a User Exit Defining how Clauses and Keywords are Formatted by Assisted Update 239
Displaying Repeating Clauses and Keywords in Assisted Update 239
Defining When Clauses and Keywords are Displayed in an Assisted Update Buffer . 240
Documenting Help for a Clause or Keyword 241
ATTRIBUTE-TYPE Syntax 241

FMT-SCREEN 247
Defining the Help for the Panel 248
Defining an MP-AID Name for a FMT-SCREEN Member 250
Defining the Panel Type 250

Defining a Point of Return for the Control Program. 250
Defining the Appearance of the Panel When Returned to From Another Panel 251
Defining Field Control Characters 251
Defining Input and Output Fields in the FMT-SCREEN Member. 254
Specifying a Relationship to ITEM Members Defining Output Fields 255
Specifying a Relationship to ITEM members Defining Input Fields 255
Defining the Processing of the Panel 255
Defining a Command Area 261
Defining the Position of the Function Key Area. 262
Defining the Allowed User Actions for the Panel 262
Defining the Position of the Message Area. 263
Defining a One-line Header 263
Defining the Layout of the Panel 263
FMT-SCREEN Syntax 264

HDS-TABLE 268
Specifying the Member Types for Generation 269
Specifying the Relationship Types for Generation 269
Specifying a Name for the Generated HDS Table 269
Specifying the RIM for Generation. 269
Specifying a Name for the Generated Translation Executive Routine 270
Including User-Defined EA Relationships in the Generation. 270
Example. 270
HDS-TABLE Syntax 270

HIERARCHY 271
Naming the MP-AID Members Generated from the RIM 271
Specifying the Entity Member Types Contained in the RIM 272
Assigning Values to Entity Member Types 273
Specifying the Relationship Member Types Included in the RIM 273
Defining Mutually Exclusive Relationship Member Types 274
Assigning Values to Relationship Member Types 275
Defining Collective Member Types 276
Specifying the User-defined Attributes Common to all Member Types 277
Assigning Parameter, Line, and Format Line Numbers to User-defined Attributes. . . 278
Specifying the UDR and UDRS Clauses to be Included in the RIM 279
HIERARCHY Syntax 279

INFOBANK-PANEL 282

ITEM 282
Defining a Title 283
Defining Lower, Upper or Mixed Case Mode 283
Defining Valid Input Values 283
Defining the Form of the Data 284
Defining Help 285
ITEM Syntax. 285

MEMBER-TYPE 286

Defining a Base or User-defined Member Type 286
Defining the Keywords With Which the Member Type is Encoded 287
Defining Keywords With Which the Member Type can be Interrogated 288
Defining Keywords That Can Be Specified in a REPORT DOWN-TO Command . . . 289
Tailoring GLOSSARY, REPORT, WHAT, and WHICH Output 289
Tailoring LIST Output 290
Tailoring SHOW UDS Output 291
Tailoring GLOSSARY and LIST Headings and Totals Output 291
Specifying Generic User-defined Attributes 292
Specifying Non-Generic User-defined Attributes 293
Defining a Member Type Level Number 294
Disallowing Relationships Between Members of the Same Member Type 294
Allowing and Disallowing Relationships Via Specified Clauses 295
Automatically Defining EA Relationships 296
Preventing a Member Type Being Displayed in the Panel Interface/Displaying IMS Collective Member Types 297
Defining Naming Conventions for Entity Members 298
Defining Complex Naming Conventions 301
Specifying the Clauses and Keywords Displayed During Assisted Update 302
Defining an Alias Identifier 302
Documenting Help for a Member Type 303
MEMBER-TYPE Syntax 303

MEMBER-TYPE-GROUP 306
Specifying the Entity Member Types Contained in the Group 306
Defining a Member Type Cluster Menu Option 307
Specifying the Member Types Selected from the Cluster Menu 308
MEMBER-TYPE-GROUP Syntax 309

RELATIONSHIP-CLASS 310
RELATIONSHIP-CLASS Syntax 311

RELATIONSHIP-GROUP 311
Defining a Group of Relationship Member Types 311
Defining Mutually Exclusive Relationship Member Types 312
RELATIONSHIP-GROUP Syntax 313

RELATIONSHIP-TYPE 314
Naming the Relationship Member Type 315
Defining the Relationship Type Class 316
Tailoring GLOSSARY, REPORT, WHAT, and WHICH Output 317
Tailoring LIST Output 317
Tailoring LIST and GLOSSARY Headings Output 318
Defining the Source and Target Member Types 318
Disallowing Unencoded Source and Target Members 319
Defining a Permitted Number of Relationships 319
Making Relationships via the Relationship Member Type Mandatory 320
Controlling the Removal of Members Participating in a Relationship 320

Allowing and Disallowing Duplicate Relationships 321
Allowing a Member to be Both the Source and Target of a Relationship 322
Documenting the Order of Retrieval of Source and Target Members 322
Specifying the User-defined Attributes that can be Included in the Member Type . . . 323
Allowing and Disallowing Relationships Via Specified Clauses 324
Automatically Defining EA Relationships 325
Defining Naming Conventions for Relationship Members 326
Taking a User Exit Defining Complex Naming Conventions 330
Preventing a Member Type Being Displayed in the Panel Interface 330
Specifying the Clauses and Keywords Displayed During Assisted Update 331
Defining an Alias Identifier 332
Documenting Help for a Member Type 332
RELATIONSHIP-TYPE Syntax. 332

SEXEC 335

10 Life Cycle Services Introduction 337

Concepts 337

Benefits 340

11 Member Types Defining a Life Cycle Model 343

12 Enabling Life Cycle Services 345

The VX, VXA, VXC, AND VXP Commands 346

13 Instructions that Produce Deliverables or Display Prerequisites 347

Macros 348
:CASE 348
:DCSTANDARD 349
Listing Members 351
:DISPLAY. 353
:LEVEL. 354
:LINE-COMMAND 354
:STANDARD 356

Commands 359
DCUPD 359
DCUPD Syntax 359
HARDCOPY. 360
MATRIX. 360
MTHELP. 362
PROJLIST. 363
PROJVIEW 366

14 An Example of a Life Cycle Model 367

An Example of a Phase 368
An Example of an Activity 369
An Example of a Subactivity 370
An Example of a LIFE-CYCLE-OBJECT-TYPE 371

15 Producing Documentation 373

16 Procedures for Creating and Maintaining Life Cycle Models 375

17 Project Management: Interactive Functions 377

Create a Project 379
Assign Existing User to Project 379
Add and Assign New User 380
Exclude User From Project 381
Delete User From Repository 381
Delete Project From Repository 381
List all Users 382
List Projects Visible From the Current Status 382
List all Projects 382
Task Management 383
Select Project 383
Monitor Task Development 385
Review of Project Members 386
Remove Dummies From Project-View 386
Include Project Related Members in Project-View 386

18 PROJECT-VIEW Members 387

Functions 387
Naming Conventions 387
IMPACT ANALYSIS & PROJECT-VIEW: MDG_PROJECTVIEW_RUCOUNTS 388

Appendix A
Listing of RIM Definition 389

Appendix B
Superseded Macros 401

:DO FOR 401
:DO FOR Syntax 402

:DO FOREVER 402

:DO WHEN 403
:DO WHEN Syntax 403

:DO WHILE 404
:DO WHILE Syntax 404

:ELSE 405

:ENDDO 405

:ENDIF 405

:IF 406
:IF Syntax 407

:LEAVE 408

:LOOP 408

Index 411

Preface

This *ASG-MethodManager Administration* describes how to tailor, generate, and administer your ASG-MethodManager (herein called MethodManager). MethodManager is a repository-based software product that enables your organization to integrate the following components into one Information Engineering (IE) environment:

- Industry standard development methodologies (such as SSADM), ASG-supplied methodologies and organization-specific working methods
- Computer Aided Software Engineering (CASE) tools provided by ASG and by other vendors
- The information your organization has stored in the repositories, catalogs, directories, libraries and/or databases supplied by ASG and other vendors.

Allen Systems Group, Inc. (ASG) provides professional support to resolve any questions or concerns regarding the installation or use of any ASG product. Telephone technical support is available around the world, 24 hours a day, 7 days a week.

ASG welcomes your comments, as a preferred or prospective customer, on this publication or on any ASG product.

About this Publication

This publication consists of these chapters:

- Chapter 1, "Introducing Administrative Roles," introduces the various administrative roles required for the successful implementation and use of MethodManager.
- Chapter 2, "MethodManager Introduction," provides an overview of the steps used by the MethodManager Administrator to generate an operational MethodManager environment.
- Chapter 3, "Repository Information Models," describes how to define and/or tailor a repository information model.
- Chapter 4, "Defining the Panel Interface," describes how to define the panel interface.

- Chapter 5, "Enabling the Environment," describes how to make the repository information model and panel interface operational.
- Chapter 6, "Customizing the Environment," describes how to customize the environment by changing the setting of ASG-supplied global variables.
- Chapter 7, "User Exits," describes the local and global exits routines provided.
- Chapter 8, "Macros," provides macros reserved for the systems administrator.
- Chapter 9, "Member Types," describes the member types defining: repository information models (RIM), panel interface, and executive routines.
- Chapter 10, "Life Cycle Services Introduction," describes the concepts and benefits of LifeCylce Services.
- Chapter 11, "Member Types Defining a Life Cycle Model," describes how to define life cycle models.
- Chapter 12, "Enabling Life Cycle Services," describes enabling LifeCycle Services.
- Chapter 13, "Instructions that Produce Deliverables or Display Prerequisites," describes a combination of instructions such as: executive routines, macros, primary and line commands, Procedures language functions, directives, and variables.
- Chapter 14, "An Example of a Life Cycle Model," provides an example of a Life Cycle Model.
- Chapter 15, "Producing Documentation," describes different way to produce documents.
- Chapter 16, "Procedures for Creating and Maintaining Life Cycle Models," describes functions with which you can create and maintain Life Cycle Models.
- Chapter 17, "Project Management: Interactive Functions," describes MethodManager's project management capabilities.
- Chapter 18, "PROJECT-VIEW Members," describes functions and naming conventions for PROJECT-VIEW members created and maintained by MethodManager.

Publication Conventions

Allen Systems Group, Inc. uses these conventions in technical publications:

Convention	Represents
ALL CAPITALS	Directory, path, file, dataset, member, database, program, command, and parameter names.
Initial Capitals on Each Word	Window, field, field group, check box, button, panel (or screen), option names, and names of keys. A plus sign (+) is inserted for key combinations (e.g., Alt+Tab).
`lowercase italic monospace`	Information that you provide according to your particular situation. For example, you would replace *`filename`* with the actual name of the file.
`Monospace`	Characters you must type exactly as they are shown. Code, JCL, file listings, or command/statement syntax. Also used for denoting brief examples in a paragraph.
Vertical Separator Bar (|) with underline	Options available with the default value underlined (e.g., Y|N).

The following conventions apply to syntax diagrams that appear in this publication.

Diagrams are read from left to right along a continuous line (the "main path"). Keywords and variables appear on, above, or below the main path.

Convention	Represents
➤➤	At the beginning of a line indicates the start of a statement.
➤◄	At the end of a line indicates the end of a statement.
———➤	At the end of a line indicates that the statement continues on the line below.
➤———	At the beginning of a line indicates that the statement continues from the line above.

Keywords are in upper-case characters. Keywords and any required punctuation characters or symbols are highlighted. Permitted truncations are not indicated.

Variables are in lower-case characters.

Statement identifiers appear on the main path of the diagram:

```
➤———COMMAND———————————————————————————————➤
```

A required keyword appears on the main path:

Convention	Represents

```
➤──────── COMMAND ──── KEYWORD ────────────────────────────────►
```

An optional keyword appears below the main path:

```
➤──────── COMMAND ──┬──────────┬──────────────────────────────►
                    └── KEYWORD ──┘
```

Where there is a choice of required keywords, the keywords appear in a vertical list; one of them is on the main path:

```
➤──────── COMMAND ──┬─ KEYWORD1 ─┬──────────────────────────►
                    ├─ KEYWORD2 ─┤
                    └─ KEYWORD3 ─┘
```

or

```
➤──────── COMMAND ──┬─ KEYWORD1 ─┬──────────────────────────►
                    ├─ KEYWORD2 ─┤
                    └─ KEYWORD3 ─┘
```

Where there is a choice of optional keywords, the keywords appear in a vertical list, below the main path:

```
➤──────── COMMAND ──┬──────────┬──────────────────────────────►
                    ├─ KEYWORD1 ─┤
                    └─ KEYWORD2 ─┘
```

The repeat symbol, <<<<<<, above a keyword or variable, or above a whole clause, indicates that the keyword, variable, or clause may be specified more than once:

```
                        <<<<<<<<
➤──────── COMMAND ──── variable ────────────────────────────►
```

A repeat symbol broken by a comma indicates that if the keyword, variable, or clause is specified more than once, a comma must separate each instance of the keyword, variable, or clause:

```
                        <<< , <<
➤──────── COMMAND ──── variable ────────────────────────────►
```

The repeat symbol above a list of keywords (one of which appears on the main path) indicates that any one or more of the keywords may be specified; at least one must be specified:

Convention	Represents

```
                          <<<<<<<<<<<<<<<<<
>------------COMMAND-------------+-KEYWORD1-+-------------------------->
                                 +-KEYWORD2-+
```

The repeat symbol above a list of keywords (all of which are below the main path) indicates that any one or more of the keywords maybe specified, but they are all optional:

```
                          <<<<<<<<<<<<<<<<<
>------------COMMAND-------------+-KEYWORD1-+-------------------------->
                                 +-KEYWORD2-+
```

1 Introducing Administrative Roles

Successful implementation and use of MethodManager requires these types of administrative roles:

- Systems administration which includes:
  - Installing and enabling MethodManager
  - Enabling repository information models
  - Global tailoring (of the character set and PF keys for example)
  - Installation and tailoring of the PGW Graphical Workbench facility
  - Creating logon profiles
  - Ensuring security and backup of the MP-AID
  - General maintenance
- Repository administration/control which includes:
  - Controlling one or more repositories
  - Controlling the repository information model (RIM) for each of those repositories
  - Ensuring security of access
  - Ensuring security of content and backup of the repository
  - Maintaining change control mechanisms (called status)
- Life cycle model maintenance which includes:
  - Building life cycle models to corporate standards
  - Applying corporate methods and proprietary methodologies via life cycle models
  - Project management which includes:
  - Assigning users to projects
  - Assigning life cycle models to projects

Systems administration. The responsibility of the MethodManager Administrator or Systems Administrator: this is likely to be someone with full-time responsibility for the smooth running of MethodManager in your organization.

Installation of MethodManager is the responsibility of the Administrator although in many organizations a different person may actually do the installation. Refer to your Manager Products installation manual for details of installing MethodManager.

Administration functions are usually performed using MethodManager command interface. However, RIM definition and enabling is also supported by the panel interface.

Documentation written specifically for Administrators is in this publication, in *ASG-ManagerView System Administrator's Tailoring,* and in *ASG-Manager Products Systems Administrator's Manual*.

Repository administration. Is the responsibility of the repository Controller: this may be the same person as the Administrator or it may be someone in Data Administration or even application development. A Repository Controller has responsibility for one (or more) repositories used by your organization.

Controller's functions are usually performed using MethodManager's command interface. Documentation describing those functions can be found in *ASG-Manager Products Controller's Manual*. This publication is primarily for Administrators, but it will be of interest to repository Controllers who are involved in the definition of RIMs.

Life Cycle Model maintenance and project management. Can be the responsibility of either Data Administrators who meet with application development project teams and/or the Managers of those development projects. Alternatively, the definition of Life Cycle Models may be the responsibility of personnel specifically employed within your organization to define and propagate working methodologies: it very much depends upon the structure of your organization.

Life Cycle Management functions enable you to define Life Cycle Models in the repository and construct the models onto the MP-AID. You can then assign the models to projects using the Project Management functions.

Project Management functions enable you to plan and control projects. You can create projects, specify who will work on them, and ensure that the development of a project follows a methodology defined in a Life Cycle Model. This is achieved using a interactive dialog that guides a user through a project and guides the project through its life cycle.

Standards. To prevent malfunctions you are recommended to ensure that global and command variable names:

- Are meaningful and at least two characters long
- Are *not* identical to any MP-AID member

2 MethodManager Introduction

As MethodManager Administrator you are responsible for installing, implementing, and successfully running MethodManager within your organization. On receiving MethodManager for the first time, or on receiving a new release of MethodManager, you have to carry out the process of generating an operational MethodManager environment (after the supplied software is installed).

The process of environment generation involves these major steps:

- Establishing the repository information model(s) that will define the type and structure of information that can be entered into the repository/repositories you use
- Customizing the panel interface supplied to compliment the repository information models you use and the standards and conventions used in your organization
- Customizing the functions provided so that they operate in the way which best suits your organization.

Chapter 3, "Repository Information Models," on page 5 describes how to define and/or tailor a repository information model, Chapter 4, "Defining the Panel Interface," on page 45 how to define the panel interface, and Chapter 5, "Enabling the Environment," on page 89 how to enable (that is, make operational) both. The subsequent chapters describe the different ways in which the operation of MethodManager can be changed to suit the way you want it to work.

ASG supplies you with a default repository information model and panel interface. Both are model-driven in that they are defined in repository members in the Administration repository, from which the run-time versions are generated.

To change what is supplied, adjust the definition in the Administration repository and then re-generate (re-enable) the run-time version. Similarly you can create your own repository information models and user interface panels by creating definitions in the Administration repository and then enabling them.

ASG strongly recommends that you create separate statuses in the Administration repository, based upon the statuses supplied, to hold tailored and new definitions. The statuses supplied are:

- MDRIM (which contains the definition of the MethodManager Dictionary/Repository Information Model)
- ADMIN (which contains the definition of the panel interface as well as definitions of the executive routines and help panels that make up the MethodManager panel interface).

By using the status facility in this way you will always be sure of which definitions are supplied by ASG. This is particularly useful when ASG provides a new release with revised versions of definitions provided in a previous release: in this situation you need to be able to evaluate the changes you made and the changes ASG made and decide how you want to deal with both (merge ASG's changes with yours to form a new definition, ignore ASG's changes etc.).

3 Repository Information Models

This chapter describes how to implement a repository information model (RIM). Implementing a RIM involves three main steps:

- Design (deciding what entity and relationship types you want)
- Definition (creating definitions in the Administration repository that represent the entity and relationship types)
- Enabling (making the RIM part of one or more production repositories)

This chapter includes these chapters:

What is a Rim? 6

Modeling the Real World 7
Entity Types and Relationship Types 8
Entities and Relationships 10
Properties of Relationship Types 10

How a RIM Models the Real World 17
EA and ER Relationship Types. 18
Naming Conventions. 21

Defining a RIM 22
UDS Member Types 24
Panel-Interface Member Types 25
HDS-TABLE Member Type. 26

Example of Implementing a RIM 26
Designing the RIM 27
Defining the RIM 29
The Enabled Environment. 32

Checklist of Steps for Implementing a RIM 36

Managing META-DATA 38
Ensuring Integrity 39
Examples of Integrity Checking 40
Removing Members 41
Interrogating a Model 43

Chapter 5, "Enabling the Environment," on page 89 covers the enabling stage, by which you make the RIM part of one or more production repositories.

What is a Rim?

A repository information model (RIM) is a model that defines the structure of the information and the relationships that can be documented in a repository.

In the context of Manager Products repositories a RIM defines:

- The types of meta-data that can be modeled in a repository
- Naming conventions for meta-data
- The structure of the panel interface
- How meta-data displays when a user updates or adds meta-data

All RIMs are fully tailorable. You can tailor the RIM supplied by ASG or you can implement a RIM from scratch.

Designing a RIM to suit all your company's needs is a complex task, but it is crucial for the successful use of a repository.

If you design a RIM with care then:

- Many aspects of the integrity of your meta-data are guaranteed.
- Users find the repository easy to use.
- Users only see the meta-data relevant to their task.

If a RIM is badly designed then it may, for example, be possible to represent information on the repository in two or more different ways. This makes the meta-data unnecessarily complicated.

If a RIM is designed with care then often information can only be represented in one way. A user or tool may try to represent it in another way but the repository will not allow it.

A RIM also defines a default panel interface. You can tailor this panel interface to suit your users. You can:

- Alter default panels
- Define your own panels

For example, you can add panels for tasks frequently performed by your users, but not catered for by the default panels.

You can export a RIM to a local repository on a PWS Graphical Workbench (PGW), in order to guarantee compatibility of host and local meta-data.

Modeling the Real World

Repositories cannot directly model the complexity and inconsistency of the real world. Instead they model an abstraction of the real world consisting of:

- Entity types
- Relationship types and their properties
- Entities
- Relationships

When you model the real world you define:

- The names of the entity and relationship types and the names and format of their attribute types
- The entity or relationship types that relationships of each type can join
- Properties of relationship types

Names and Format. Your model might for example consist of two entity types person and program and one relationship type codes. You can specify the attribute types for the two entity types and the relationship type. For example, the relationship type might have an attribute type operating-system, since if a program exists for two or more operating systems the same person may not be coding all the versions.

Relationships. If there were no restrictions on the entity or relationship types that relationships of each type can join then, for example, any entity could be joined to any other entity by a relationship of any type. Such freedom could quickly lead to chaos in the repository.

Instead of allowing chaos you can define precisely which entity or relationship types each relationship type can join. These restrictions help maintain the integrity of models held in the repository. For example, you might define that a relationship of type codes can only join a person entity to a program entity.

Properties. The properties of relationship types put additional restrictions on relationships of that type. For example, you can define a relationship type *codes* so that a program cannot be modeled on the repository unless the person coding it is also modeled. That is, you have decided that meta-data about programs is useless unless meta-data about the programmers who coded them is also recorded.

Entity Types and Relationship Types

An *entity type* defines a group of entities with the same attributes. For example, the entity type program defines the characteristics of programs.

Each entity type can have a *naming convention*. This specifies the range of strings from which the name of an entity of this type can be chosen. For example, you can specify that an instance of entity type program must begin with the string PR-.

In RIM diagrams an entity type is shown as a box. Figure 1 shows the entity types, person, program, and subroutine. For example, using such a RIM you can model a person LAF or a program DMC28.

Figure 1 • Entity Types

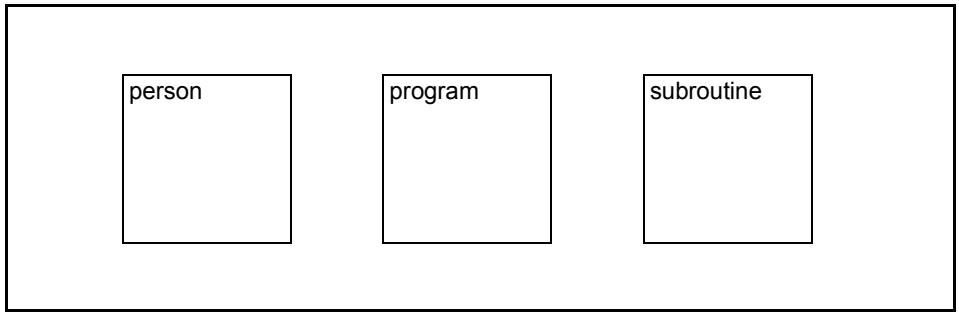
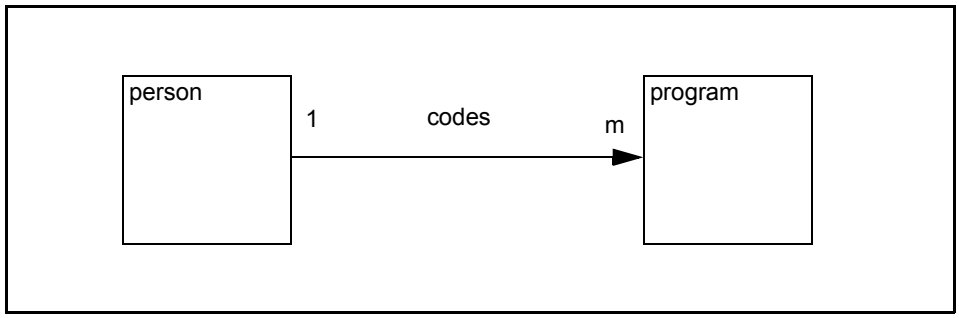


Figure 2 • Relationship Types



A *relationship type* defines a group of relationships with the same attribute types. For example, a relationship type codes, defining a group of relationships between people and programs. That is, person codes program. This is illustrated in Figure 2. For example, using such a RIM you can model the fact that the person LAF is coding the program DMC28. (For the moment ignore the 1 and the m in the diagram. Their meaning is described later in this chapter.)

A relationship type, like an entity type, can have a naming convention.

In Figure 2 the relationship type connects just two entity types. However, in general a relationship type connects a group of entity or relationship types (the source) to another such group (the target). For example, the target of a relationship type codes might be the two entity types program and subroutine. The meaning of this construct must be described in terms of relationships and so is deferred to the next section.

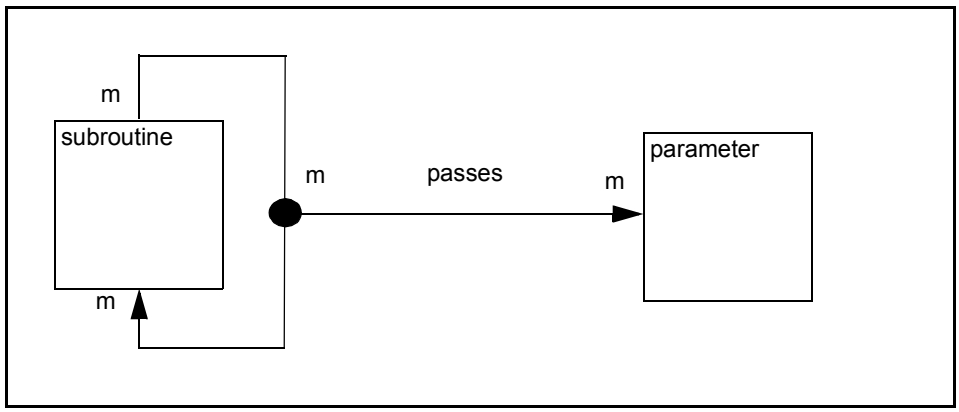
If source and target each have one element the relationship type is called *simple*. Almost all the relationship types described in this chapter are simple.

A simple relationship type can by definition connect any of these:

- An entity type to an entity type (including an entity type to itself)
- A relationship type to a relationship type (including a relationship type to itself)
- An entity type to a relationship type, and vice versa

Figure 3 shows a relationship type passes connecting a relationship type calls to an entity type parameter. For example, using such a RIM you can model the fact that subroutine open Window calls subroutine displayMenu passing an integer array. Such a construct contains a *relationship on a relationship*.

Figure 3 • Relationship Types on a Relationship Type



Relationship types have a primary direction and a inverse direction with respectively the names:

- Primary name
- Inverse name

To avoid clutter, RIM diagrams only show the primary direction (shown by the arrow) and the primary name. However, the two directions have equal precedence. For example, you can interrogate in either direction.

In Figure 2 on page 8 the relationship type's primary name is codes. The inverse name might be coded-by.

In relationship types the source and target are specific to the direction, that is:

- The target of the primary direction is the source of the inverse direction.
- The source of the primary direction is the target of the inverse direction

For example, in Figure 3:

- The source of passes in the primary direction is calls.
- The target of passes in the inverse direction is calls.

Relationship types have properties to help you define the semantics of your repository information. These are discussed in "Properties of Relationship Types" on page 10.

Entities and Relationships

An *entity* is an instance of the entity type to which it belongs. For example, the program DMS14 is an instance of the entity type program.

A *relationship* is an instance of the relationship type to which it belongs. For example, the fact that the program DMS14 calls the program DMC28 is modeled by a relationship of type *calls*. DMS14 is the source of the relationship and DMC28 its target.

A relationship has a single source and a single target.

Now return to the question of what it means for the source or target of a relationship type to be a group of entity or relationship types. Let R be a relationship type. A relationship of type R has as its source an entity or relationship of any type from R's source. Similarly, a relationship of type R has as its target an entity or relationship of any type from R's target.

An example will make this clear. If the target of a relationship type *codes* is the group of entity types program and subroutine then the target of a relationship of type *codes* can be an entity of type program or subroutine.

Properties of Relationship Types

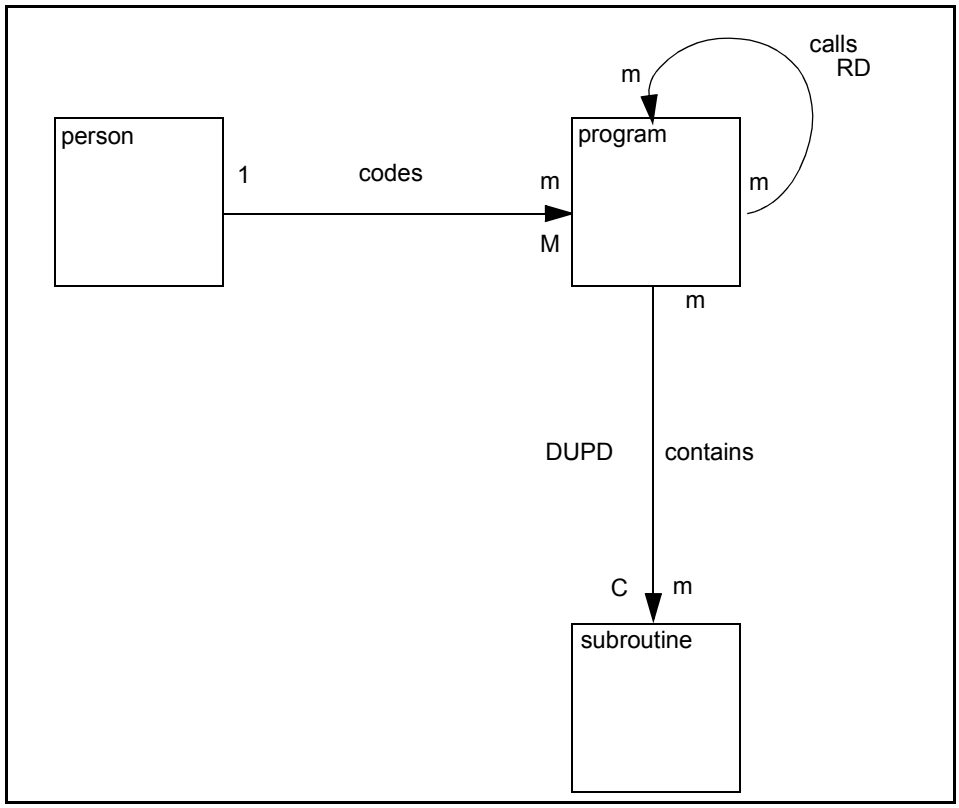
Relationship types have six properties to help you define the semantics of your repository information. The repository uses these to maintain the integrity of your meta-data. These properties are:

- Cardinality
- Mandatory
- Controlled
- No-dummies
- Duplicates
- Recursive

Each property is independent of the others. The first four of these properties are defined separately for source and target. The last two apply to the whole relationship type. In RIM diagrams properties are marked on the relationship types using letters or numbers.

The properties are defined below, with examples. The examples are mostly taken from the simple program development environment shown in Figure 4. A full description of this environment is given later in this chapter ("ER Relationship Types" on page 21).

Figure 4 • Simple Program Development Environment



The Cardinality Property

The cardinality property specifies how many times an entity can participate in relationships of a particular type. The full definition is given below.

Suppose p and q are both positive integers and p is greater than or equal to q.

If the source of a relationship type has a *cardinality of q,p,* then all instances of the target type must be the target of zero or between q and p instances of that relationship type.

The same is true for the target. That is, if the target of a relationship type has a cardinality of q,p, then all instances of the source type must be the source of zero or between q and p instances of that relationship type.

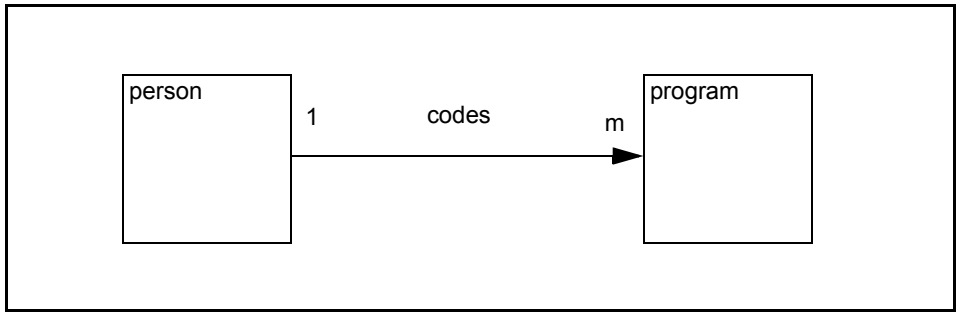
Cardinality of p is short for cardinality of 1,p. The default cardinality is an unrestricted cardinality, a cardinality of many.

For example, you can specify in a relationship type Codes that a person can code many programs, but that a program can only be coded by one person. For example, using such a RIM you can model the fact that person LAF is coding DMC28 and DMS14 but you cannot model the fact that person LAF and person TJM are both coding DMC28.

In RIM diagrams cardinality is given, using the letter m (for many) or positive integers, at each end of the relationship type. Cardinality is always shown in full, unlike other properties which have defaults. A cardinality of many is shown by the letter m.

Figure 5 gives an example of the cardinality property. The source of relationship type *codes* has a cardinality of 1 (that is 1,1) and the target has a cardinality of many (that is 1,many).

Figure 5 • Cardinality Property



The Mandatory Property

The mandatory property specifies if an entity must participate in a relationship of a particular type. The full definition is given below.

If the source of a relationship type is *mandatory* then an instance of that source type must be the source of an instance of that relationship type.

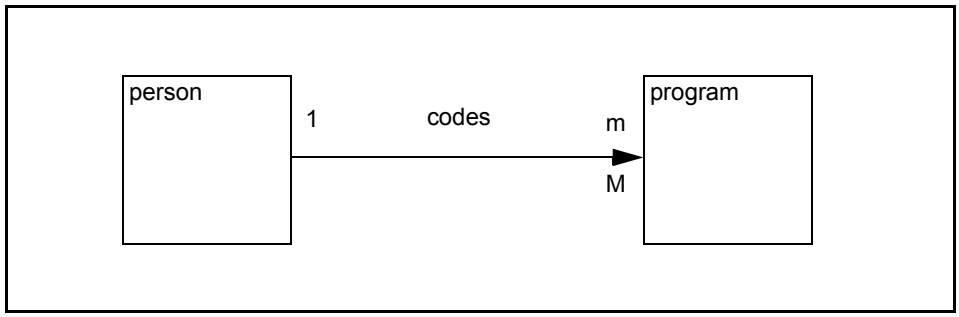
The same is true for the target. That is, if the target of a relationship type is mandatory then an instance of that target type must be the target of an instance of that relationship type.

For example, you can specify in a relationship type codes that a program can only be modeled if the person coding it is also modeled.

In RIM diagrams mandatory is shown by the letter M. Non-mandatory (that is, optional) is shown by the letter O. The O for optional can be omitted.

Figure 6 gives an example of the mandatory property. The target of relationship type codes is mandatory, the source is optional.

Figure 6 • Mandatory Property



The Controlled Property

The controlled property specifies if extra action takes place when a relationship is removed. The full definition is given below.

Suppose an instance i is the source of precisely one relationship r of a particular type. If the source of this relationship type is *controlled*, then when r is removed from the repository i also is removed. In such a case, removal of the target removes the relationship, which removes the source.

If the source of the relationship type is not controlled it is *uncontrolled*. If the source of the relationship type is uncontrolled then removing the relationship (if allowed) has no effect on the source.

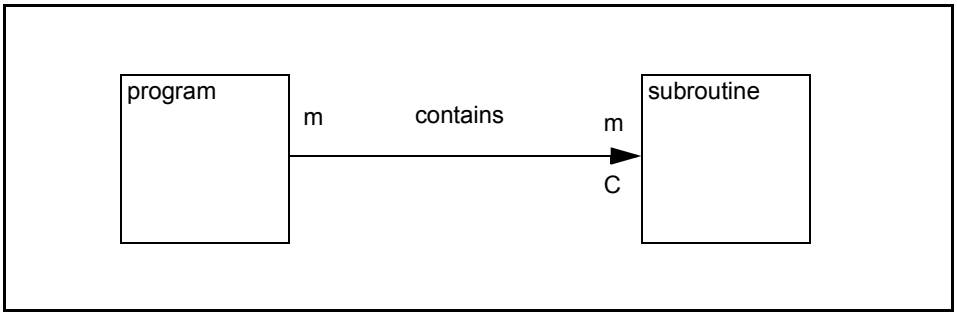
The same is true for the target. Suppose an instance i is the target of precisely one relationship r of a particular type. If the target of this relationship type is controlled, then when r is removed from the repository i also is removed. In such a case, removal of the source removes the relationship, which removes the target.

If the target of the relationship type is not controlled it is uncontrolled. If the target of the relationship type is uncontrolled then removing the relationship has no effect on the target.

For example, in a relationship type Contains you can in effect specify that a subroutine should only be modeled while a program that contains it is modeled.

In RIM diagrams controlled is shown by the letter C. Uncontrolled is shown by the letter U. U for uncontrolled can be omitted. Figure 7 gives an example of the controlled property. The target of relationship type contains is controlled.

Figure 7 • Controlled Property



The Dummies Property

The dummies property specifies if a relationship can be created before its participant entities or relationships exist. The full definition is given below. (It is called the dummies property because in a repository a reference to an undefined member creates a dummy member.)

If the source of a relationship type is *dummies-disallowed* then an instance of that type can only be created when the source exists. If the source of a relationship type is not dummies-disallowed then it is *dummies-allowed*.

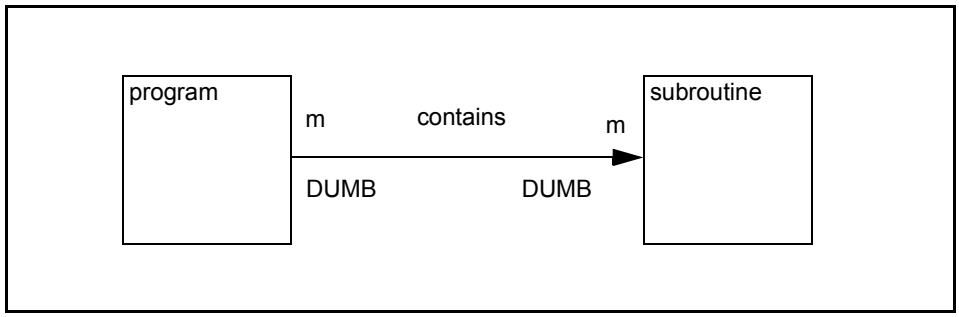
The same is true for the target. That is, if the target of a relationship type is dummies-disallowed then an instance of that type can only be created when the target exists. If the target of a relationship type is not dummies-disallowed then it is dummies-allowed.

For example, you can specify that a relationship of type contains can only be modeled when both the program and subroutine are already modeled. (The alternative would be to just give, in the relationship, the names of the program and subroutine, and model them later.)

In RIM diagrams dummies-allowed is shown by the letters DUMA. Dummies-disallowed is shown by the letters DUMD. DUMA for dummies-allowed can be omitted.

Figure 8 gives an example of the dummies property. The source and target of relationship type contains are both dummies-disallowed.

Figure 8 • Dummies Property



The Duplicates Property

The duplicates property specifies if duplicate relationships are allowed. The full definition is given below.

Two relationships are *duplicates* if all the following are true:

- They are instances of the same relationship type
- Their sources are the same
- Their targets are the same

For any relationship type you can specify any one of the following:

- Duplicates allowed
- Duplicates disallowed
- Duplicates only allowed if distinguished by a particular attribute.

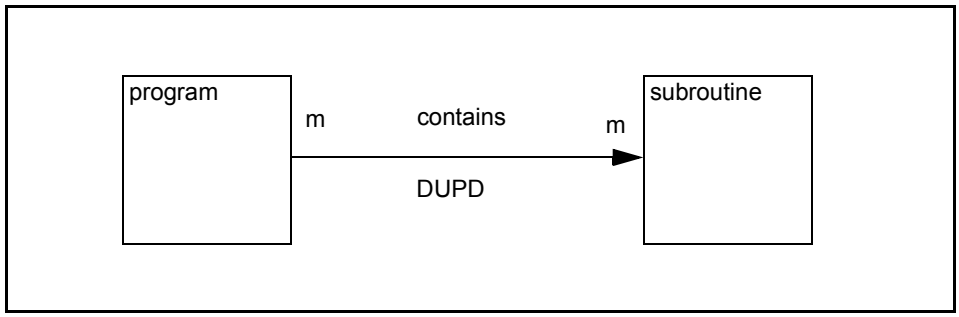
A relationship type is *duplicates-allowed* if duplicates are allowed. If a relationship type is not duplicates-allowed it is *duplicates-disallowed*.

For example, in a relationship type codes you can specify that a person coding a program more than once cannot be modeled. For example, with such a RIM the person LAF cannot be recorded as coding DMC28 twice.

In RIM diagrams duplicates-allowed is shown by the letter DUPA. Duplicates-disallowed is shown by the letters DUPD. DUPA for duplicates-allowed can be omitted.

Figure 9 gives an example of the duplicates property. Relationship type contains is duplicates-disallowed.

Figure 9 • Duplicates Property



The Recursive Property

The recursive property specifies if recursive relationships are allowed. The full definition is given below.

If the intersection of a relationship type's source and target is not null then it is possible for an instance of that relationship type to have source and target equal. (Remember that, for relationship types, source and target are sets of entity and/or relationship types.) This can be restated as follows: if the source and target of a relationship type have an entity type or relationship type in common then it is possible for an instance of that relationship type to have source and target the same.

A relationship with source and target the same is recursive.

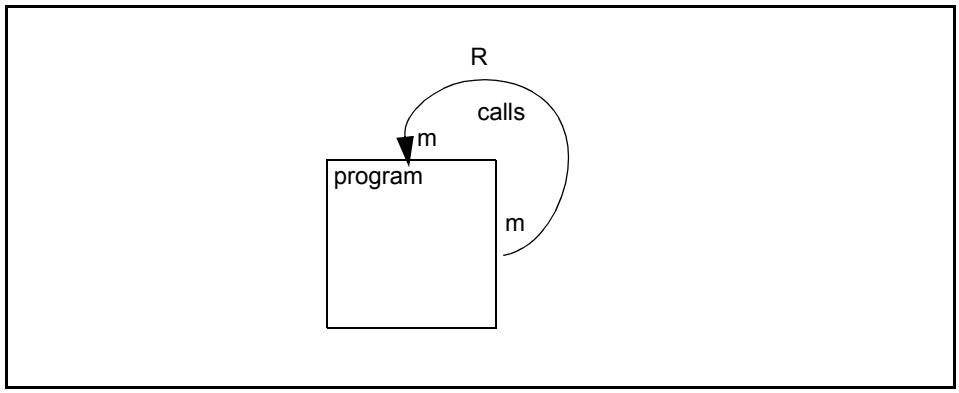
For any relationship type you can allow or disallow recursive relationships. A relationship type is *recursive* if recursion is allowed. If a relationship type is not recursive it is *recursion-disallowed*.

For example, in a relationship type Calls you can specify that a program can call itself. For example, with such a RIM you can record that DMC28 calls itself.

In RIM diagrams recursive is shown by the letter R. Recursion-disallowed is shown by the letters RD. R for recursive can be omitted.

Figure 10 gives an example of the recursive property. Relationship type Calls is recursive.

Figure 10 • Recursive Property



How a RIM Models the Real World

This section describes how a Manager Products repository represents the real world. It assumes that you have a basic knowledge of Manager Products. If you do not have this basic knowledge then you may have to refer at times to the *ASG-MethodManager User's Guide*.

An entity type is represented by an *entity member type*. For example, an entity type Program might be represented by an entity member type PROGRAM.

For further details of defining entity member types refer to "MEMBER-TYPE" on page 286.

An entity is represented by an *entity member*. For example, a program DMS14 might be represented by an entity member PR-DMS14 of type PROGRAM.

There are two types of relationship type in a Manager Products repository:

- Entity-relationship (ER) relationship types
- Entity-association (EA) relationship types

An *ER relationship type* is represented by a *relationship member type*. For example, a relationship type codes might be represented by a relationship member type CODES.

For further details of defining relationship member types refer to "RELATIONSHIP-TYPE" on page 314.

An *EA relationship type* is represented by a relationship clause type in a member type. For example, the CALLS relationship clause type in the PROGRAM member type represents an EA relationship type calls.

Each entity or relationship member type has a *naming convention* representing the naming convention of the corresponding entity or relationship type.

A *member type* is an entity or relationship member type.

An *attribute type*, in an entity type or ER relationship type, is represented by a non-relationship clause type in a member type.

Since there are two types of relationship type in a Manager Products repository, there are also two types of relationship:

- Entity-relationship (ER) relationships
- Entity-association (EA) relationships

An *ER relationship* is represented by a relationship member. For example, an ER relationship of type calls might be represented by a relationship member of type CALLS.

An *EA relationship* is represented by a relationship clause in a member. For example, an EA relationship of type calls might be represented by a CALLS clause in a PROGRAM member.

An *attribute*, in an entity or relationship, is represented by a non-relationship clause in a member. For example, an attribute of type *parameter-number* in a relationship of type *passing* might be represented by an integer clause of type PARAMETER-NUMBER in a relationship member of type PASSING.

EA and ER Relationship Types

ER relationship types can have attribute types and properties, whereas EA relationship types cannot. However, EA relationship types are easier to maintain and more efficient when you encode or interrogate members. You should use them if you want to record the existence of a relationship and nothing more. For example, in strategic information planning.

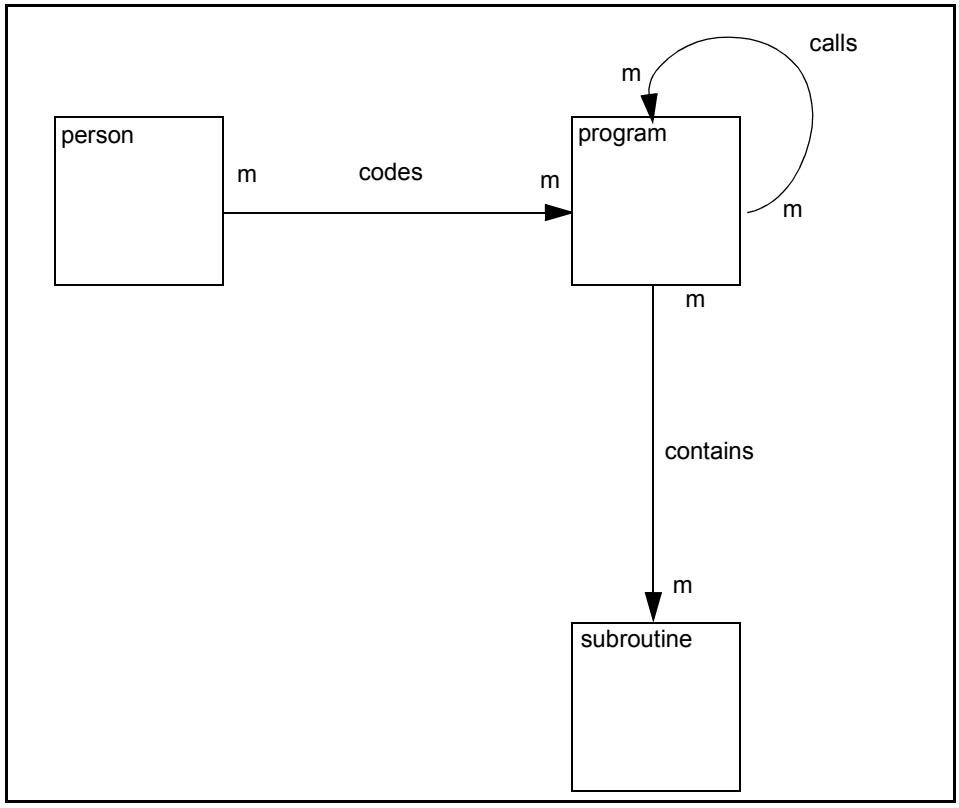
EA relationship types do not have properties, but they can be considered to have fixed settings of some properties. For example, the so-called user-defined relationships (UDRs), which you can use to define your own EA relationship types, can be considered to have a source and target cardinality of many.

In a RIM you can:

- Use both ER and EA relationship types
- Use only ER relationship types
- Use only EA relationship types

The examples in the next two sections make the distinction between the two types of relationship type clearer.

Figure 11 • Example Using EA Relationship Types



EA Relationship Types

Consider the RIM in Figure 11. The relationship types in it are EA relationship types. It has these entity types:

- Person
- Program
- Subroutine

All entities must be of one of these three types.

The RIM also has these relationship types:

- Codes, which can only join person to program
- Calls, which can only join program to program
- Contains, which can only join program to subroutine

All the relationships must be of one of these three types.

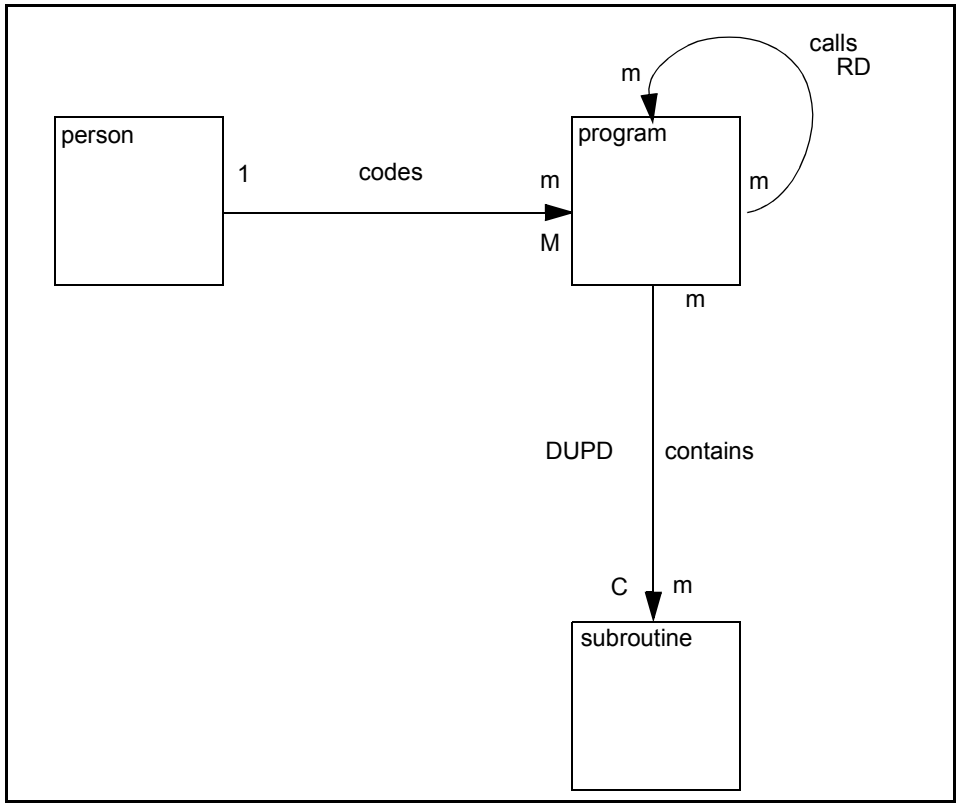
Relationships can only be used in the allowed ways. For example, with this RIM it is impossible to directly join person to subroutine. To do this you might extend the relationship type codes so that person codes program and person codes subroutine.

So the above gives the design. But how to implement it? The ASG-defined EA relationship types calls and contains are many-to-many, so you can use them for the calls and contains relationship types. To create the codes EA relationship type rename a user-defined relationship (UDR). A UDR is also a many-to-many relationship type.

Note:

Codes should be a 1-to-many relationship type. It can only be defined as such using an ER relationship type.

Figure 12 • Example Using ER Relationship Types



ER Relationship Types

Consider the RIM in Figure 12 on page 20. This is the same as the RIM in Figure 11 on page 19 except that:

- The cardinality of codes is now 1-to-many, instead of many-to-many
- The relationship types have properties other than cardinality

The restrictions on the entities and relationships that can be modeled on the repository are as follows:

- All the restrictions in the RIM in Figure 11 on page 19
- Additional restrictions defined by ER relationship types

The additional restrictions include the following:

- A program cannot be modeled unless the person coding it is also modeled.
- When a program is removed any subroutines only used by it are also removed.
- Two or more people cannot code the same program.

These additional restrictions are defined using respectively the Mandatory, Controlled, and Cardinality properties:

Naming Conventions

Consistent names are essential for the efficient management of your meta-data. ASG recommends that you maintain a company list of standard keywords with required abbreviations, which you can use as the prefix and/or suffix of member names. This reduces the possibility of homonyms (same name, different thing) and synonyms (same thing, different name). Each of the member types ASG provides have default naming conventions, which you can tailor to your requirements.

You specify the naming convention for a particular member type in the NAMING clause of the relevant MEMBER-TYPE or RELATIONSHIP-TYPE member.

In the NAMING clause you can specify compulsory characters that must appear in a member name, and wildcard characters, which allow for permutations in names. You can use the numeric wildcard character to specify that a particular part of a name must be a numeric character. You can also specify the minimum, maximum, or exact length of a name. If a user enters a member name that does not conform to your convention, the name is rejected.

You can use the NAMING-EXIT clause in a MEMBER-TYPE or RELATIONSHIP-TYPE member to define more complex naming conventions. You specify in the NAMING-EXIT clause the name of the executive routine that will perform the name checking; this is known as a *local exit*, because it is specific to a particular member type. For example, an executive routine can check whether the last character of the member name is a digit between 1 and 5.

You can also define *global exits* which perform name checking for all member types.

Refer to Chapter 7, "User Exits," on page 171 for details of how to define local or global exits.

You can use the :NAMKO macro to check naming conventions for members and member types.

Refer to "NAMKO" on page 201 for details of the :NAMKO macro.

You can relax the naming convention restrictions on all or specific member types, by changing the setting of the variables MDG_NAM_OLD and MDG_NAM_OLD_MEM.

Refer to Chapter 6, "Customizing the Environment," on page 103 for details of these variables.

Defining a RIM

This section gives an overview of the member types you use to define a RIM. Full specifications of these member types are given in Chapter 9, "Member Types," on page 215.

You define a RIM in your Administration repository using the following member types:

- User defined syntax (UDS) member types
- Panel-interface member types

The UDS member types define the entity and relationship member types in your production repository. The panel-interface member types can be used to alter the panel interface for the Administration repository or production repositories.

The *UDS member types* are: HIERARCHY, MEMBER-TYPE-GROUP, MEMBER-TYPE, ATTRIBUTE-GROUP, ATTRIBUTE-TYPE, RELATIONSHIP-GROUP, RELATIONSHIP-TYPE, and RELATIONSHIP-CLASS.

Figure 13 • Relationship Types Between UDS Member Types

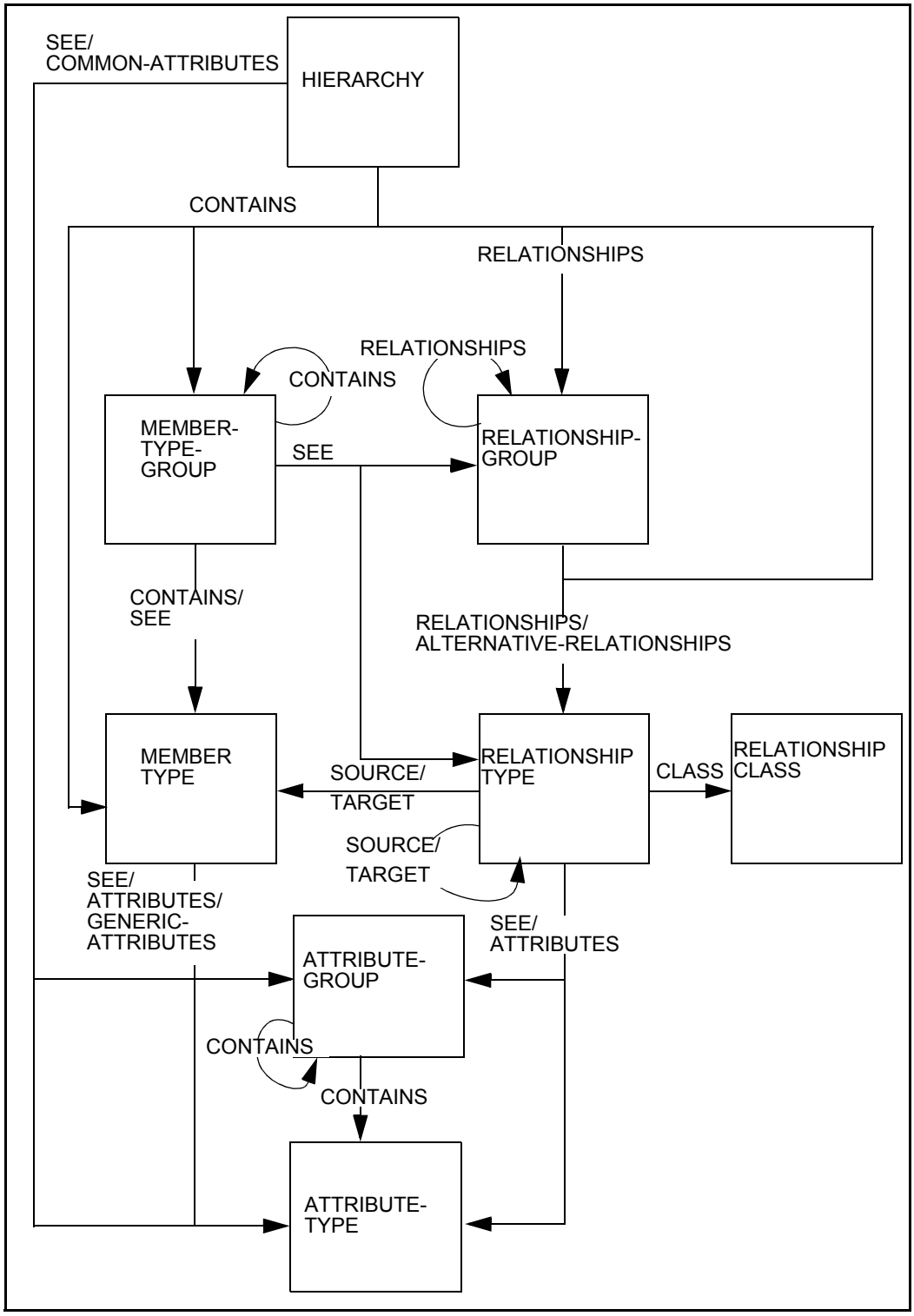


Figure 13 on page 23 shows the relationship types between these member types. For example, the member types allowed as the source of a relationship member type are specified by filling in the SOURCE clause of the appropriate RELATIONSHIP-TYPE member.

Members of these types must be defined before you can enable your RIM.

The *panel-interface member types* are as follows:

- INFOBANK-PANEL
- SEXEC
- ITEM
- EXECUTIVE-ROUTINE
- FMT-SCREEN

Members of these types are supplied by ASG in the Administration repository. You alter and/or copy them in order to alter the panel interface for the Administration repository or production repositories.

To export all or part of a RIM to a local repository you:

- Define a HDS-TABLE member
- Execute the enable HDS-TABLE function

UDS Member Types

The HIERARCHY member draws all the members defining the RIM together and gives information such as the name of the RIM.

A MEMBER-TYPE-GROUP member defines a group of entity member types, and can optionally define menus, known as cluster menus, listing groups of member types.

A MEMBER-TYPE member defines:

- An entity member type (representing an entity type)
- Naming conventions for entity members of this type
- How an entity member of this type is displayed in an assisted update buffer
- Help information for the entity member type

An ATTRIBUTE-GROUP member defines a group of clause types.

An ATTRIBUTE-TYPE member defines:

- A clause type
- Help for the clause type

A RELATIONSHIP-GROUP member defines a group of ER relationship types.

A RELATIONSHIP-TYPE member defines:

- A relationship member type (representing a relationship type)
- Naming conventions for relationship members of this type
- How a relationship member of this type is displayed in an assisted update buffer
- Help information for the relationship member type
- Which relationship-type classes the relationship member type belongs to

A RELATIONSHIP-CLASS member defines a relationship-type class. You define relationship-type classes in order to simplify interrogations.

Panel-Interface Member Types

A FMT-SCREEN member defines one of these panels in the panel interface:

- Menu
- List
- Input
- Output

For further details of these panel types refer to *ASG-MethodManager User's Guide*.

An INFOBANK-PANEL member defines help available with a panel defined by a FMT-SCREEN member.

SEXEC and EXECUTIVE-ROUTINE members define the processing that takes place when data is entered in a given field of a panel defined by a FMT-SCREEN member.

In this context, an ITEM member is used in defining an input or output field in a panel defined by a FMT-SCREEN member.

Refer to Chapter 4, "Defining the Panel Interface," on page 45 for full details of defining the panel interface.

HDS-TABLE Member Type

A HDS-TABLE member gives information such as:

- The name of the RIM to be exported
- The required member types

You must define a HDS-TABLE member before you enable a HDS table. Refer to Chapter 5, "Enabling the Environment," on page 89 for details of enabling HDS tables.

Example of Implementing a RIM

This section gives a complete example of implementing a RIM. It covers:

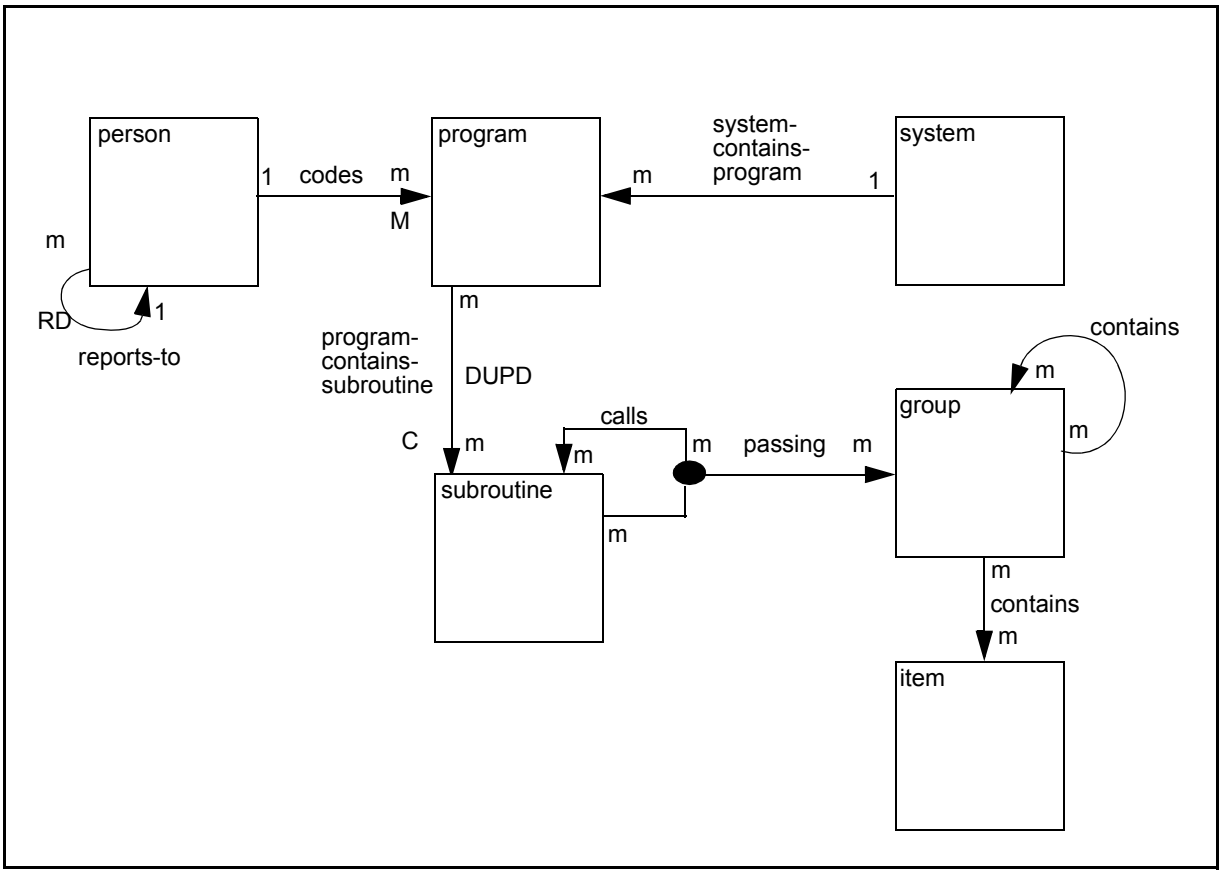
- Designing the RIM
- Defining the RIM
- Enabling the RIM onto a production repository

In full the steps followed are those given in "Checklist of Steps for Implementing a RIM" on page 36.

Note:

Member definitions in this section are given as part of REPLACE commands. If you execute these REPLACE commands the members are added to your repository.

Figure 14 • Example RIM



Designing the RIM

Figure 14 on page 27 shows a design for a program development environment. Such a model would normally be a submodel of a RIM, but here it is the whole RIM. The design has these entity types:

- Person
- Program
- System
- Subroutine
- Group
- Item

All entities must be of one of these six types.

The design has these relationship types:

- Reports-to, representing a person reporting to a person
- Codes, representing a person coding a program
- System-contains-program, representing programs grouped together into systems
- Program-contains-subroutine, representing subroutines grouped together to form programs
- Calls representing a subroutine calling a subroutine
- Passing, representing the passing of one or more parameters when a subroutine is called
- Contains, representing the structure of parameters

All relationships must be of one of these seven types.

Each entity type and ER relationship type has a naming convention.

Let the naming conventions for each entity type be simply the first two letters of the entity type's name, followed by a hyphen, followed by any sequence of characters.

Let the naming conventions for each ER relationship type be simply the first two or three letters of the relationship type's name, followed by a hyphen, followed by any sequence of digits.

Note these points about the RIM:

- A person cannot report to more than one person, a person cannot report to himself or herself
- A program must be coded by precisely one person
- A program cannot be contained in more than one system
- A program cannot contain a subroutine more than once
- When a subroutine calls a subroutine the parameters can be recorded and also the order in which they must be passed
- A subroutine is only modeled while any program containing it is also modeled

Relationship types are implemented as either ER relationship types or EA relationship types.

Relationship types must be implemented as ER relationship types if:

- They have properties other than cardinality
- The source and target cardinalities are not both many
- They have attributes

So all the above relationship types except Contains must be ER relationship types. Contains can be either EA or ER. ASG recommends that in such a case you choose EA. So let Contains be EA.

Defining the RIM

Refer to these specific sections when defining the RIM:

"Defining the RIM" on page 29 describes the members that define the example RIM.

"Defining Entity Types" on page 30 describes the definitions of the entity types.

"Defining Relationship Types" on page 31 describes the definitions of the relationship types.

Appendix A,"Listing of RIM Definition," on page 389 gives a full listing of the members that define the RIM.

"The Enabled Environment" on page 32 describes the enabled environment.

For convenience in interrogations a relationship-type class Contains is defined by:

```
REPLACE CONTAINS.
RELATIONSHIP-CLASS
PRIMARY-NAME CONTAINS
INVERSE-NAME CONTAINED-BY
.
```

The relationship-type class Contains consists of these relationship types:

- system-contains-program
- program-contains-subroutine

Defining Entity Types

Each entity type is represented by an entity member type, defined by a MEMBER-TYPE member. For example, the entity type Program might be defined by:

Figure 15 • Defining MEMBER-TYPE member

```
REPLACE MMT-PROGRAM.
MEMBER-TYPE
IS PROGRAM
ALIAS "PR"
NAMING "PR-**"
STANDARD-LITERAL "PROGRAM"
ENCODE-KEYWORD PROGRAM
SEE MAT-LANGUAGE
  , MAT-DATE-WRITTEN
RELATIONSHIPS VIA UDR1 DISALLOWED
RELATIONSHIPS VIA UDR2 DISALLOWED
RELATIONSHIPS VIA UDR3 DISALLOWED
RELATIONSHIPS VIA UDR4 DISALLOWED
RELATIONSHIPS VIA UDR5 DISALLOWED
RELATIONSHIPS VIA UDR6 DISALLOWED
RELATIONSHIPS VIA UDR7 DISALLOWED
RELATIONSHIPS VIA UDR8 DISALLOWED
RELATIONSHIPS VIA UDR9 DISALLOWED
RELATIONSHIPS VIA UDRS DISALLOWED
RELATIONSHIPS VIA SEE DISALLOWED
RELATIONSHIPS VIA INPUTS DISALLOWED
RELATIONSHIPS VIA OUTPUTS DISALLOWED
RELATIONSHIPS VIA UPDATES DISALLOWED
RELATIONSHIPS VIA PARAMETERS DISALLOWED
RELATIONSHIPS VIA PASSING DISALLOWED
RELATIONSHIPS VIA CONTAINS DISALLOWED
RELATIONSHIPS VIA CALLS DISALLOWED
RELATIONSHIPS VIA QUALIFIED-ON DISALLOWED
RELATIONSHIPS VIA ACCESS DISALLOWED
RELATIONSHIPS VIA GIVING DISALLOWED
RELATIONSHIPS VIA GIVING-THROUGH DISALLOWED
RELATIONSHIPS VIA GIVING-(THROUGH) DISALLOWED
RELATIONSHIPS VIA EDIT-NAME DISALLOWED
RELATIONSHIPS VIA COUNTS-AS DISALLOWED
RELATIONSHIPS VIA SELECTING DISALLOWED
RELATIONSHIPS VIA USER-PASSWORD DISALLOWED
RELATIONSHIPS VIA GIVING-IN DISALLOWED
RELATIONSHIPS VIA COMMBLOCK-MEMBER DISALLOWED
RELATIONSHIPS VIA SOURCE-SSR DISALLOWED
RELATIONSHIPS VIA TARGET-SSR DISALLOWED
RELATIONSHIPS VIA VIEWS DISALLOWED
HELP
A PROGRAM member documents a set of actions or instructions that a machine is
capable of executing as a whole.
.
```

Most of the definition simply consists of disallowing the default EA relationship types. The example RIM disallows all EA relationship types for every entity type except Group.

Each user-defined entity member type inherits clause types from an ASG-supplied member type via the BASED-ON or IS clause types of the MEMBER-TYPE member type. For example, the PROGRAM entity member type inherits the clause types of the ASG-supplied PROGRAM entity member type. Inherited relationship clause types can be disallowed, inherited non-relationship clause types cannot be disallowed.

All entity types are needed except Person. In reality entity type Person needs extra attribute types. You represent extra attribute types using the ATTRIBUTES clause type of the MEMBER-TYPE member type.

All non-disallowed inherited clause types and user-defined clause types are available when you update a member using the command interface. In assisted update buffers only the clause types specified in the SEE clause of the MEMBER-TYPE member defining that member type are displayed. For example, in assisted update buffers for PROGRAM members only two of the inherited clause types display.

Defining Relationship Types

Each ER relationship type is represented by a relationship member type, defined by a RELATIONSHIP-TYPE member. For example, the ER relationship type Passing might be defined by:

Figure 16 • Defining RELATIONSHIP-TYPE member

```
REPLACE MRT-S-CALLS-S-PASSING-GROUP.
RELATIONSHIP-TYPE
PRIMARY-NAME PASSING
INVERSE-NAME PASSED-BY
ALIAS "PA"
NAMING "PA-#>"
SHORT-LITERAL "PASSING"
SOURCE TYPE MRT-SUB-CALLS-SUB
   CARDINALITY MANY
TARGET TYPE MMT-GROUP
   CARDINALITY MANY
ATTRIBUTES MAT-PARAMETER-NUMBER
SEE MAT-SOURCE
  , MAT-TARGET
  , MAT-PARAMETER-NUMBER
RELATIONSHIPS VIA UDR1 DISALLOWED
RELATIONSHIPS VIA UDR2 DISALLOWED
RELATIONSHIPS VIA UDR3 DISALLOWED
RELATIONSHIPS VIA UDR4 DISALLOWED
RELATIONSHIPS VIA UDR5 DISALLOWED
RELATIONSHIPS VIA UDR6 DISALLOWED
RELATIONSHIPS VIA UDR7 DISALLOWED
RELATIONSHIPS VIA UDR8 DISALLOWED
RELATIONSHIPS VIA UDR9 DISALLOWED
RELATIONSHIPS VIA UDRS DISALLOWED
RELATIONSHIPS VIA SEE DISALLOWED
HELP
A PASSING member documents a subroutine passing a parameter to a subroutine by
defining a source CALLS member and a target GROUP member.
.
```

This relationship type has a user-defined clause type specified by the ATTRIBUTES clause.

The SEE clause specifies that only the given three of the available clause types are to be displayed in assisted update buffers.

The relationship type Contains is represented by the relationship clause type CONTAINS in the GROUP member type.

The Enabled Environment

Panel Interface

The member type cluster menu, shown in Figure 17, has a single option, since only one MEMBER-TYPE-GROUP member is directly referenced by the CONTAINS clause of the HIERARCHY member.

When you choose this option the selection list panel shown in Figure 18 on page 33 displays. This has one line for every member type in our RIM.

Figure 18 on page 33 shows an AUPD command entered against a new member CA-001 of type CALLS. Figure 19 on page 33 shows the assisted update buffer when this command is executed. In order to create the member CA-001 the user must fill in the SOURCE and TARGET clauses.

Figure 19 on page 33 shows the PF1 help for the CALLS member type. General help for the member is listed, followed by help for each of the two clause types.

Figure 17 • Member-Type Cluster Menu

```
Host MANAGER Products Direct Access
Edit  Transfer  PF-Keys  Special-Keys  Options  Colors  Host-Commands  Help
TD50000                        METHODMANAGER                ToolSet SERVICES

===>

              Update         Create and Update Members

select one of the following

           1  EXAMPLE         MODELING PROGRAM DEVELOPMENT









F1=Help F3=Return F4=Exit                                                  MS
```

Figure 18 • Selection List Panel

```
Host MANAGER Products Direct Access
Edit  Transfer  PF-Keys  Special-Keys  Options  Colors  Host-Commands  Help
TD51000                        METHODMANAGER                ToolSet SERVICES

===>

               EXAMPLE       MODELING PROGRAM DEVELOPMENT

 ----> Enter in the Line Command Area
        s = member list  aupd = update a member

sel.  Member                          Type                          M/R/C
===== PE-**                           PERSON                          M
===== PR-**                           PROGRAM                         M
===== SY-**                           SYSTEM                          M
===== SU-**                           SUBROUTINE                      M
===== GR-**                           GROUP                           M
===== IT-**                           ITEM                            M
=AUPD CA-001                          CALLS                           R
===== PA-#>                           PASSING                         R
===== RE-#>                           REPORTS-TO                      R
===== CO-#>                           CODES                           R
===== SCP-#>                          SYSTEM-CONTAINS-PROGRAM         R
===== PCS-#>                          PROGRAM-CONTAINS-SUBROUTINE     R

F1=Help F3=Return F4=Exit F5=Refresh F7=Backward F8=Forward                MS
```

Figure 19 • Assisted Update Buffer

```
Host MANAGER Products Direct Access
Edit  Transfer  PF-Keys  Special-Keys  Options  Colors  Host-Commands  Help
DICTIONARY UPDATE:CA-001                          LINE: 00000 OF 00008 WAITIN
DICTIONARY:MMR                                          VIEW: 001 TO 073 >>>
====>
===== *** TOP OF DATA ***
=====  ------------------------------------------------------------------------
=====  *CALLS                                                              *CA
=====  ------------------------------------------------------------------------
=====  *SOURCE                Source of relationship
===== ??.
=====  ------------------------------------------------------------------------
=====  *TARGET                Target of relationship
===== ??.
===== *** END OF DATA ***
```

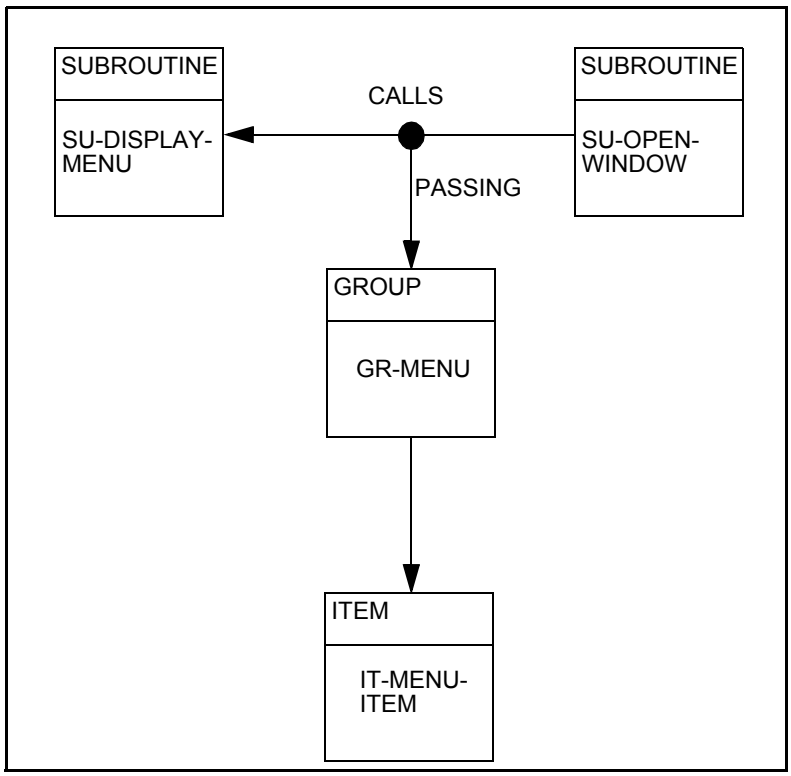
Figure 20 • Member-Type Help

```
Host MANAGER Products Direct Access
Edit  Transfer  PF-Keys  Special-Keys  Options  Colors  Host-Commands  Help
HELP                          METHODMANAGER                          XTCATEST

====> SELECT
       Valid Commands In UPDATE Buffer:
         CANCEL   Leave update buffer, cancel updates
         ACCEPT   Leave update buffer with SAVE (without ENCODE)
         F3/F15   Function keys, save (as RETURN)
         ADIS     Display empty skeleton for edited member type
         HELP     Line command, shows HELP text specific to attribute,
                  enter in the line containing the attribute name
       HELP for Member Type

        CALLS
         A CALLS member documents a subroutine calling a subroutine by
         defining a source and a target SUBROUTINE member.
        SOURCE
         The name of the member which is the source of the relationship.
        TARGET
         The name of the member which is the target of the relationship.

F1=Help F3=Return F4=Exit F5=Refresh F7=Backward F8=Forward
                                                                    9103
```

Meta-data

Figure 21 shows a simple example of meta-data that can be held in our enabled environment. The meta-data represents the subroutine openWindow calling the subroutine displayMenu and passing parameters of a particular type. The source of the passing relationship is itself a relationship. Such a structure is called a relationship on a relationship.

Figure 21 • Sample Meta-data



The full listing of the meta-data is as follows:

Figure 22 • Meta-data

```
REPLACE SU-DISPLAY-MENU.
SUBROUTINE
LANGUAGE
"ASSEMBLER"
DATE-WRITTEN
"21.10.92"
.
REPLACE SU-OPEN-WINDOW.
SUBROUTINE
LANGUAGE
"PL/1"
DATE-WRITTEN
"21.10.92"
.
REPLACE CA-001.
CALLS
SOURCE
SU-OPEN-WINDOW
TARGET
SU-DISPLAY-MENU
.
REPLACE PA-001.
PASSING
SOURCE
CA-001
TARGET
GR-MENU
PARAMETER-NUMBER
1
.
REPLACE GR-MENU.
GROUP
CONTAINS
IT-MENU-ITEM
.
REPLACE IT-MENU-ITEM.
ITEM
REPORTED-AS
1 ALPHANUMERIC   10
.
```

Checklist of Steps for Implementing a RIM

This section gives a checklist of steps for implementing a RIM. There are three main stages in the steps:

- Design
- Definition
- Enabling

You can also, if you wish, tailor the panel interface.

The steps are as follows:

1 Design your RIM. This consists of two steps:

- Decide what entity types and relationship types you want to model. What attribute types do each entity and relationship type have?
- Decide on naming conventions for each entity type and relationship type.

Refer to "Modeling the Real World" on page 7 for further details.

2 Log on to your Administration repository

3 Start up ToolSet SERVICES. If ToolSet SERVICES does not start automatically, enter:

```
TSS ;
```

4 Enter Administration functions (option 8 on the main menu).

5 For each attribute type, create an ATTRIBUTE-TYPE member (if there isn't already a suitable one amongst those supplied by ASG).

Refer to the ATTRIBUTE-TYPE specification for further details.

6 If in your RIM design you find that several entity or relationship types have the same group of attribute types, it will simplify your RIM definition if you create an ATTRIBUTE-GROUP member.

Refer to the ATTRIBUTE-GROUP specification for further details.

7 For each entity type, create a MEMBER-TYPE member (if there is not already a suitable one). As well as defining the entity type this member defines how an entity of that type is displayed in assisted update buffers.

You may wish to disallow any unwanted EA references from this entity type. EA references are disallowed using the RELATIONSHIPS VIA clause type in member type MEMBER-TYPE.

Refer to "MEMBER-TYPE" on page 286 further details.

8 If in your RIM design you several times refer to the same group of entity types, it will simplify your RIM definition if you create a MEMBER-TYPE-GROUP. For example, if two ER relationship types have the same group of entity types as their source or target, you would define one MEMBER-TYPE-GROUP member and reference it twice, rather than repeat the list in both the RELATIONSHIP-TYPE members.

Refer to "MEMBER-TYPE-GROUP" on page 306 for further details.

9 For each relationship type with properties, create a RELATIONSHIP-TYPE member (if there isn't already a suitable one). This is called an ER relationship type. The member also defines how a relationship of that type displays in assisted update buffers.

You may wish to disallow any unwanted EA references from the ER relationship type. EA references are disallowed using the RELATIONSHIPS VIA clause type in member type RELATIONSHIP-TYPE.

Refer to "RELATIONSHIP-TYPE" on page 314 for further details.

10 For each relationship type without properties, you have a choice, since a relationship type without properties can be defined by a relationship clause type in one of the participating entity member types (this is called an EA relationship type) or by a RELATIONSHIP-TYPE member (this is called an ER relationship type). For details of which type you should choose refer to "How a RIM Models the Real World" on page 17.

If you have defined an ER relationship type you may wish to disallow any unwanted EA references from it. EA references are disallowed using the RELATIONSHIPS VIA clause type in member type RELATIONSHIP-TYPE.

Refer to "RELATIONSHIP-TYPE" on page 314 for further details.

11 If in your RIM design you several times refer to the same group of ER relationship types, it will simplify your RIM definition if you define a RELATIONSHIP-GROUP member.

Refer to "RELATIONSHIP-GROUP" on page 311 for further details.

12 To simplify interrogations in your production repository you can define a relationship-type class by creating a RELATIONSHIP-CLASS member.

Refer to "RELATIONSHIP-CLASS" on page 310 for further details.

13 Create a MEMBER-TYPE-GROUP member for each selection list panel that you want to define. You may be able to use some or all of the MEMBER-TYPE-GROUP members you defined in earlier steps. These selection list panels appear as options on the member-type cluster menu, which is displayed in ToolSet SERVICES in the production repository whenever you need to select a member.

Refer to "MEMBER-TYPE-GROUP" on page 306 for further details.

14 Create a HIERARCHY member.

Refer to "HIERARCHY" on page 271 for further details.

15 Enable the RIM onto the production repository. Refer to "Enabling the Environment" on page 89 for further details. You may wish to examine the output from a SHOW UDS command to check that you have defined the RIM correctly.

16 Download all or part of the RIM to local repositories as necessary. Refer to *ASG-ManagerView Administration Guide* for further details.

For details of tailoring the panel interface refer to Chapter 4, "Defining the Panel Interface," on page 45.

To change parts of the RIM other than the panel interface you:

- Alter the members in the Administration repository defining that RIM
- Re-enable the RIM

It may not be necessary to redo all the rules. For further details of re-enabling refer to Chapter 5, "Enabling the Environment," on page 89.

Managing META-DATA

This section describes how the entity-relationship facilities affect how you manage meta-data in a production repository. For full details of managing meta-data you should also refer to *ASG-Manager Products Dictionary/Repository User's Guide*.

Integrity rules defined in the repository information model (RIM) help you ensure that the models in the repository are meaningful. (Do not confuse these rules with those by which you enable a RIM.) Before using a model, for example printing, interrogating or exporting it, you may wish to check that it passes these rules. See "Ensuring Integrity" on page 39 for an overview of this checking process and "Examples of Integrity Checking" on page 40 for some practical examples.

The rules defined in the RIM control not only what can be stored in the repository but also what happens when members are removed from the repository. This is described in "Removing Members" on page 41.

Finally, "Interrogating a Model" on page 43 gives an overview of interrogation.

Ensuring Integrity

This section describes how you ensure that a model passes the integrity rules defined in the RIM.

Encoded members are in one of two states:

- Check-ok
- Check-needed

If you wish to distinguish between check-ok and check-needed members in list output then set the check-character using the SET CHECK-CHARACTER command. Refer to *ASG-ControlManager User's Guide* for further details of the SET CHECK-CHARACTER command.

A model is *check-ok* if all the members in it are marked as check-ok.

Integrity checking is the process by which a model or member is checked to see if it satisfies the rules defined in the RIM.

Automatic checking is checking that takes place automatically when you create or alter a member. The alteration to the member is rejected if the checking fails.

To apply automatic checking, use the CHECK prefix command. Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for further details of the CHECK prefix command.

If you always use automatic checking then any model is by definition always check-ok: at any time you can immediately use the model without doing any checks. Thus you should, wherever possible, use automatic checking.

At any time you can *validate* a member using the VALIDATE command. When you validate a member one of the following takes place:

- The member is marked as check-ok if it passes the checks
- The member is marked as check-needed if it fails the checks, and the reasons for failure are listed

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for further details of the VALIDATE command.

It is not always possible to use automatic checking. For example, if an ER relationship type specifies that dummies are not allowed, you may have to add two or more members (not a single member) before a model is check-ok. If you validate the model after adding a single member, the model might not be check-ok.

Because of this the AUPD command does not use automatic checking. Instead it encodes the member and then validates it. Thus if the member fails any checks you can correct the errors, or ignore them until you need to validate the whole model.

If you alter a model and do not use automatic checking then impacted members are merely marked as check-needed. Effectively the checking is postponed until you choose to do it.

This means that if you do not always use automatic checking a model may not be check-ok. To make a model check-ok, follow the steps below:

1. Validate all the check-needed members in the model. If there are no check-needed members, stop.
2. Alter remaining check-needed members to remove the given reasons for failure.
3. Repeat from step 1.

Examples of Integrity Checking

Consider the RIM shown in Figure 14 on page 27.

It is assumed that you are adding members without automatic checking and then validating them.

If you try to model a program without modeling the person coding it, then this message is output:

```
DM00335E name DOES NOT HAVE MANDATORY CODES RELATIONSHIP
IN status-name
```

where *name* is the name of a PROGRAM member. This construct is not allowed because the target of codes is mandatory. To correct this, model the program and the person coding it.

If you try to model a person reporting to two or more people, this message is output:

```
DM00331E name VIOLATES CARDINALITY RULE FOR REPORTS-TO
IN status-name
```

where *name* is the name of a PERSON member. This construct is not allowed because the target cardinality of *reports-to* is 1. To correct this, remove one of the *reports-to* relationships.

If you model a person reporting to themselves then this message is output:

```
DM00330E name VIOLATES RECURSION RULE FOR REPORTS-TO IN
STATUS status-name
```

where *name* is the name of a REPORTS-TO member. This construct is not allowed because reports-to is recursion-disallowed. To correct this, model the person reporting to someone else, or to no one.

If you twice model a subroutine as contained-by a given program, this message is output:

```
DM00332E PROGRAM-CONTAINS-SUBROUTINE name1 DUPLICATES
name2 IN STATUS status-name
```

where *name1* and *name2* are both members of type PROGRAM-CONTAINS-SUBROUTINE. This construct is not allowed because program-contains-subroutine is duplicates-disallowed. To correct this, remove one of the duplicate relationships.

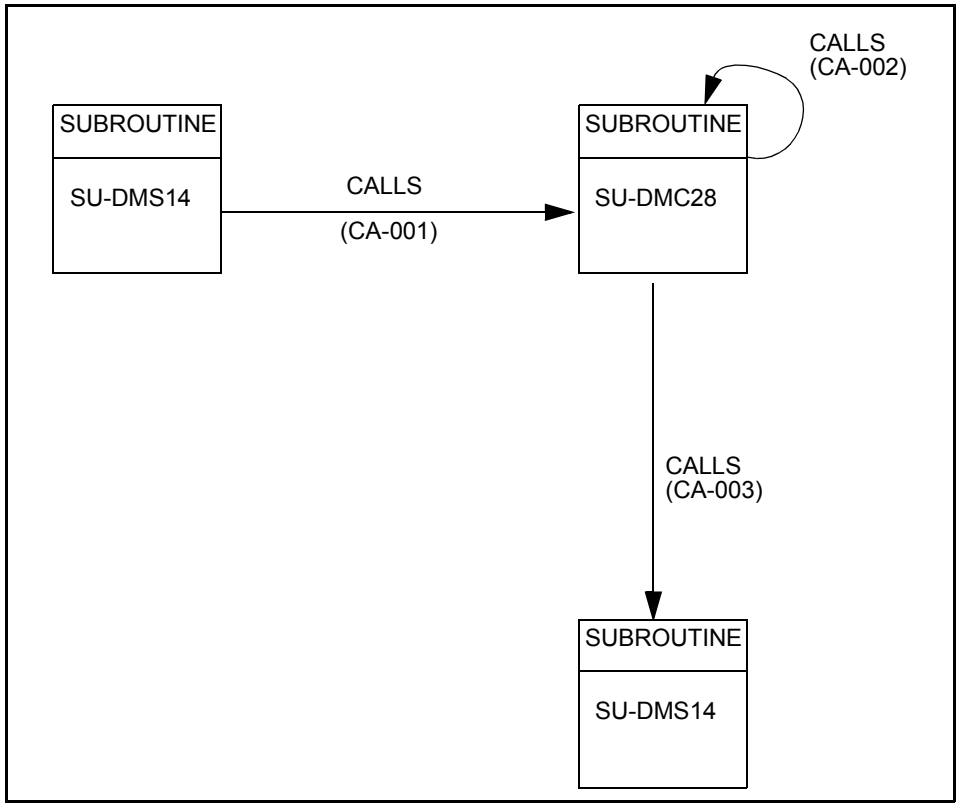
Removing Members

When you remove a member, relationships involving it are also removed. When a relationship is removed controlled participants may also be removed. Thus if you remove a single member, removals can cascade through the model.

For example, consider the meta-data diagram in Figure 23 on page 42. The names in parentheses beneath the relationship types are the names of the relationships. For example, the member defining the relationship between SU-DMC28 and SU-DMI04 is CA-003. Suppose, for this example only, that the source and target of relationship type Calls are controlled. If you remove the entity member SU-DMS14 then the members are removed in the following order until none remain:

```
CA-001
SU-DMC28
CA-002 and CA-003
SU-DMI04
```

Figure 23 • Removing Members



The mandatory property can prohibit the removal of relationships, since it specifies that a member of a particular type must participate in a relationship of a particular type. For example, in the RIM in Figure 14 on page 27 you will not be able to remove a given relationship of type CODES if doing so would leave a PROGRAM member without the mandatory relationship of type CODES. This example can be restated in terms of commands as follows:

- A CHECK REMOVE command to remove such a relationship is rejected.
- A REMOVE command without the CHECK prefix does remove the relationship and the lack of integrity is only discovered later.

The cardinality property can prohibit the removal of relationships. For example suppose the RIM in Figure 14 on page 27 specified that a person who supervises must supervise between two and five people (that is, the source cardinality of *reports-to* is 2,5). If a person supervises two people then you cannot remove just one of the *reports-to* relationships. You can, however, remove both relationships since the target of *reports-to* is optional.

Interrogating a Model

Before interrogating a model, you may have to check that the model is check-ok.

Consider the RIM in Figure 14 on page 27. Having created a model you can ask these questions:

- Which subroutines call a particular subroutine
- Which subroutines are called by a particular subroutine
- Which programs is a particular person coding
- Which subroutines are contained-by a particular program

The commands to ask these questions are as follows:

- WHICH SUBROUTINES DIRECTLY USE subroutine-name RELATED VIA CALLS
- WHICH SUBROUTINES DIRECTLY CONSTITUTE subroutine-name RELATED VIA CALLS
- WHICH PROGRAMS DIRECTLY CONSTITUTE person-name RELATED VIA CODES
- WHICH SUBROUTINES DIRECTLY CONSTITUTE program-name RELATED VIA PROGRAM-CONTAINS-SUBROUTINE

You can interrogate in the primary or inverse directions of any relationship, though in all the above commands the primary verb is specified. For example, to find the person who is coding a particular program, enter:

```
WHICH PERSON DIRECTLY CONSTITUTES program-name VIA
CODED-BY ;
```

if CODED-BY is the inverse verb of CODES. The RIM specifies that (in a check-ok model) there is always precisely one such person.

To ask more complicated questions, such as which subroutines are contained by programs that a particular person is coding, you use several interrogations or write executive routines. So for example to answer the above question, enter:

```
KEEP WHICH PROGRAMS CONSTITUTE person-name RELATED VIA
         CODES ;
PERFORM "ALSO KEEP WHICH SUBROUTINES CONSTITUTE *
             RELATED VIA
             PROGRAM-CONTAINS-SUBROUTINE "
             KEPT CLEAR-KEPT-DATA ;
```

The answer is held in the unnamed KEPT-DATA list.

A *relationship-type class* is a class of ER relationship types. The systems administrator defines relationship-type classes. Using relationship-type classes you can in one command interrogate on a range of ER relationship types.

Relationship-type classes can be used to make certain questions easier to ask. For example, suppose you want to know the subroutines contained-by programs within a given system. Since the ER relationship types *system-contains-program* and *program-contains-subroutine* belong to the relationship-type class *contains* the single command:

```
WHICH SUBROUTINES INDIRECTLY CONSTITUTE system-name
            RELATED VIA CONTAINS ;
```

answers the question. The command is executed as follows:

- From the given system it follows all relationships of types belonging to relationship-type class *Contains*. In fact all these relationships must be of type *system-contains-program*
- From the resultant PROGRAM members it then follows all relationships of types belonging to relationship-type class *contains*. In fact all these relationships must be of type *program-contains-subroutine.*
- Since there are then no relationships to follow it stops, returning all the SUBROUTINE members it found.

Note:

The EA relationship type Contains does not belong to the relationship-type class Contains, but it will be interrogated on in the above command if you omit the RELATED keyword.

For further details of interrogations refer to *ASG-Manager Products Dictionary/Repository User's Guide*.

If you wish to collect additional information during the interrogation then you can do it using DACCESSes and DRETRIEVEs via an executive routine. For example, you may wish in some interrogations to check that all members examined are check-ok. (Interrogation commands do not check the integrity state of members.)

For further details of executive routines refer to *ASG-Manager Products Procedure Language*.

4 Defining the Panel Interface

This chapter includes these sections:

Defining Panels of Different Types 48
Menus 48
Input Panels 51
List Panels 55
Output Panels 59

Tailoring Panels of the Update Cycle 61
Using Assisted Update on Views 63
Example of Using a View 64
MMRVIEW Command 65
AUPDATE Command 67

Defining Help 68
Defining Extended Help in INFOBANK-PANEL or FMT-SCREEN Members 69
Defining Extended Help in Any Member Type Other Than INFOBANK-PANEL or FMT-SCREEN 72
Defining Contextual Help in ITEM Members 75
Defining Contextual Help in Any Member Type of the Repository 80

Checking the Layout of a Defined Panel 82

Enabling a Defined Panel 87
CX Command Syntax 87

The panel interface is a combination of different panel types. ASG supplies:

- Menus
- List panels
- Input panels
- Output panels
- Help panels
- Assisted update skeletons

The panel interface is defined in these member types:

- FMT-SCREEN
- ITEM
- INFOBANK-PANEL

Use the FMT-SCREEN member type to define menus, list panels, input panels and output panels. For details of the FMT-SCREEN member type, refer to Chapter 9, "Member Types," on page 215.

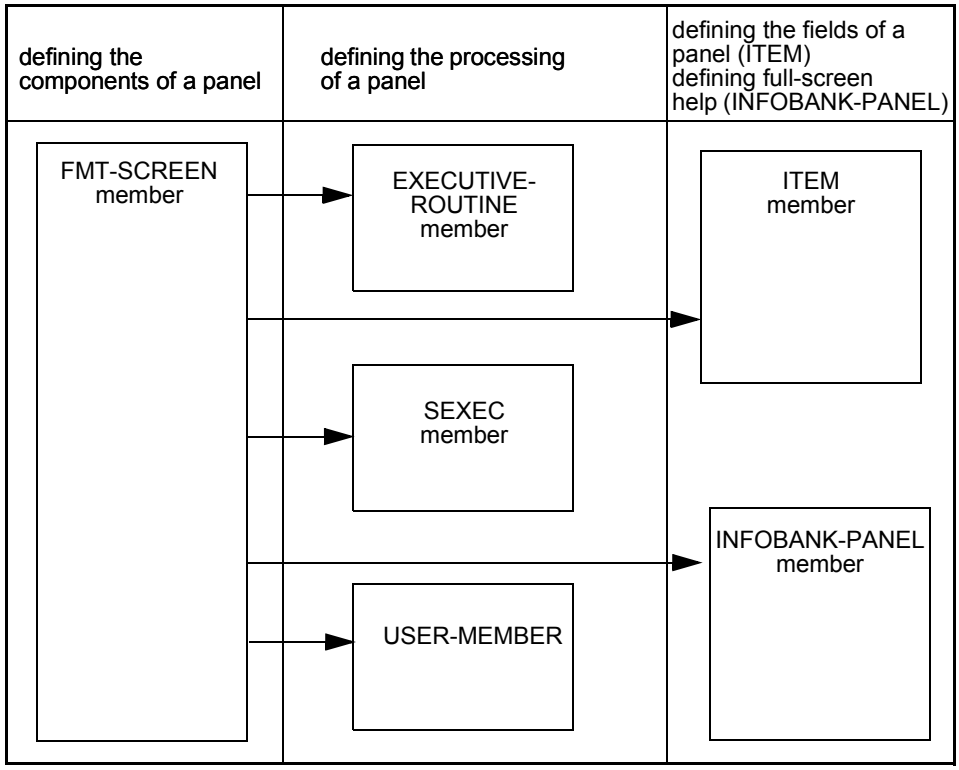
Use the ITEM member type to define input and output field of a panel. ITEM members are used as global variables by FMT-SCREEN members to specify the fields of a panel. For details of the ITEM member type, refer to Chapter 9, "Member Types," on page 215.

Use the INFOBANK-PANEL member type to define help text for an entire panel. INFOBANK-PANEL members are used by FMT-SCREEN members to supply extended help for menus, list panels, input and output panels.

Use the EXECUTIVE-ROUTINE, the SEXEC or the USER-MEMBER member type to define processes coded in Manager Products procedures language. These members are used by FMT-SCREEN members to specify the processing of a panel.

Figure 24 illustrates the relationships between the different member types.

Figure 24 • Member Types Defining the Panel Interface and its Processing



The panel interface becomes active once the repository information model (RIM) has been defined and successfully enabled.

You may want to tailor the panel interface, for instance by adding options to an existing menu to integrate tools from other vendors into ToolSet SERVICES (TSS).

To add an option to a menu, change the definition of the FMT-SCREEN member defining the menu. If the option calls an input panel not supplied by ASG, define a new input panel using the FMT-SCREEN member type.

Use the list panel TD51000 to update an existing FMT-SCREEN member and to create a new FMT-SCREEN member defining a panel.

Please note that in the panel interface the member name of a panel is displayed without its prefix SC- in the upper left-hand corner.

Defining Panels of Different Types

The following sections describe how to define menus, list panels, input and output panels using the FMT-SCREEN member type. Please note that the FMT-SCREEN members defining the panels are not shown as displayed in assisted update but as stored in the repository.

For details of the FMT-SCREEN and the ITEM member type, refer to Chapter 9, "Member Types," on page 215.

Menus

Figure 25 shows a typical example of a menu. It presents different functions for data modeling and design. To select a function enter its option in the command area.

Figure 25 • Example of a Menu

```
W00000                        METHODMANAGER           ToolSet SERVICES

===>

           Data Modeling      Data modeling and design functions

 select one of the following

            1  Member Type    List members by member type
            2  Manipulate     Workbench manipulation
            3  Analyze        Analyze workbench
            4  Generate       Generate from workbench
            5  Update         Create and update members
            6  Graphic        Graphic display of conceptual schema

            U  User           User defined option
            T  Tutorial       Data modeling and design tutorial



 F1=Help F3=Return F4=Exit
```

The menu is defined in the FMT-SCREEN member SC-W00000 shown in Figure 26.

Figure 26 • Definition of a Menu

```
FMT-SCREEN
SEE W00000H FOR 'HELP'
TYPE MENU
DECLARE-FIELDS
'IN MDG_MENU_SELECTION1 ALPHANUMERIC 1'
'OUT MDG_MESSAGE_AREA ALPHANUMERIC 70'
FUNCTION-KEY 'BOTTOM'
MESSAGE 'BOTTOM'
HEADER
'         ??HPMETHODMANAGER??XP                ToolSet SERVICES'
OPTION NEW
CALLS SC-TW10000 AT '1'
CALLS SC-W20000 AT '2'
CALLS SC-W30000 AT '3'
CALLS SC-W40000 AT '4'
CALLS SC-TW50000  AT '5'
CALLS SC-W60000 AT '6'
CALLS SC-WU00000 AT 'U'
CALLS MPEAT0000 AT 'T PASSING WT00000'
CALLS MPEAN0000 AT 'NOINPUT MESSAGE 43023 I'
CALLS MPEAN0000 AT 'NOTFOUND MESSAGE 43001 E'
CALLS MPEAN0000 AT 'NOTPGM MESSAGE 48000 E'
CONTENTS
??XP
??XP          Data Modeling      ??HPData modeling and design functions
??XP
??XP
??XPselect one of the following
??XP
??XP    ??HU_??HP   ??HP1??XP Member Type    List members by member type
??XP          ??HP2??XP Manipulate     Workbench manipulation
??XP          ??HP3??XP Analyze        Analyze workbench
??XP          ??HP4??XP Generate       Generate from workbench
??XP          ??HP5??XP Update         Create and update members
??XP          ??HP6??XP Graphic        Graphic display of conceptual
??XP
??XP          ??HPU??XP User           User defined option
??XP          ??HPT??XP Tutorial       Data modeling and design
```

The FMT-SCREEN member SC-TW00000 shown in Figure 26 on page 49 defines the menu as follows:

- The SEE clause specifies extended help for the menu. The help is to be generated from the INFOBANK-PANEL member W00000H. For details of how to define help, refer to "Defining Help" on page 68.
- No MP-AID name is specified, because the repository name of the FMT-SCREEN member defining the menu is not longer than ten characters.
- The MENU keyword defining the TYPE clause specifies that a menu is to be generated.
- The DECLARE-FIELDS clause defines an input and an output field for the menu. The position and the type (unprotected) of the input field is indicated by the field control character ??HU in the CONTENTS clause. You need not indicate the position of the output field for the message area by using field control characters in the CONTENTS clause. This is done automatically.
- The command area is not defined in the COMMAND-LINE clause but activated by the global variables MDG_MMR_CX_CMD_LINE and MDG_MMR_CX_CMD_TYPE for all menus in the environment. If you change the setting of the global variables a command area is only generated if the COMMAND-LINE clause is specified in the FMT-SCREEN member defining the menu.

  Note:

  The display of the input field defined in the DECLARE-FIELDS clause and indicated in the CONTENTS clause is suppressed in the menu (see Figure 25 on page 48) because a command area exists for the menu. If a command area is neither defined in the FMT-SCREEN member nor activated by the global variables the input field displayS automatically so that you can select an option from the menu.

- The FUNCTION-KEY and the MESSAGE clause define the position of the function key area and of the message area respectively. Both are positioned to the bottom of the panel because they are defined as BOTTOM.
- The HEADER clause defines the standard heading for the panel. Please note that field control characters can also be used in the definition.
- OPTION NEW indicates that any previously entered option will be deleted when you return to the menu from another panel.
- No strings are defined for the various field control characters, so their defaults will be used.
- The CALLS clause defines the panels which displayS when you select an option from the menu. Please note that the options specified in the CALLS clause must be identical with the options specified in the CONTENTS clause of the FMT-SCREEN member.

The last three CALLS clauses define error conditions and their corresponding messages.

- The CONTENTS clause defines the layout of the menu, using static text and field control characters. Several field control characters can be entered in one line. A field control character is active until another field control character is found in the same line. Please note that even blank lines must be protected using the relevant control characters.

Input Panels

Figure 27 shows a typical example of an input panel. Make the relevant entries in the unprotected input fields, indicated by underscores (_).

Figure 27 • Example of an Input Panel

```
TW33000                     METHODMANAGER              ToolSet SERVICES

===>

              Entity         Create Userview or Entity Report

 selection ----> ____________________________________________
 entities  ----> _ (any non-blank character selects)
 userviews ----> _           "

Use this panel to display the dependencies that can be derived
from the selected objects.









F1=Help F3=Return F4=Exit F5=Refresh
```

The input panel is defined in the FMT-SCREEN member SC-W33000 shown in Figure 28.

Figure 28 • Definition of an Input Panel

```
FMT-SCREEN
TYPE INPUT
OPTION REUSE
HEADER
'         ??HPMETHODMANAGER??XP                ToolSet SERVICES'
FUNCTION-KEY 'BOTTOM'
MESSAGE 'BOTTOM'
SEE W33000H FOR 'HELP'
DECLARE-FIELDS
'IN MDG_INPUT_TW33000_1 ALPHANUMERIC 45'
'IN MDG_INPUT_TW33000_2 ALPHANUMERIC 1'
'IN MDG_INPUT_TW33000_3 ALPHANUMERIC 1'
'OUT MDG_MESSAGE_AREA ALPHANUMERIC 79'
CALLS EX-TW33000 AT 'COMBIN YYN'
CALLS EX-TW33000 AT 'COMBIN YNY'
CALLS MPEAN0000 AT 'NOINPUT MESSAGE 43023 I'
CALLS MPEAN0000 AT 'NOTFOUND MESSAGE 43001 I'
CALLS MPEAN0000 AT 'NOTPGM MESSAGE 00110 E'
CALLS MPEAN0000 AT 'NOCOMBIN MESSAGE 43080 E'
CONTENTS
??XP
??XP              Entity        ??HPCreate Userview or Entity Report??XP
??XP
??XP selection ---->??HU??XP
??XP entities  ---->??HU ??XP (any non-blank character selects)
??XP userviews ---->??HU ??XP          "
??XP
??XPUse this panel to display the dependencies that can be derived
??XPfrom the selected objects.??XP
```

The FMT-SCREEN member SC-W33000 shown in Figure 28 on page 52 defines the input panel as follows:

- The INPUT keyword defining the TYPE clause specifies that an input panel is to be generated.
- OPTION REUSE indicates that the panel displays as it was originally, that is showing previous entries made when you return to the input panel from another panel.
- The HEADER clause defines the standard heading for the panel. Please note that field control characters can also be used in the definition.
- The FUNCTION-KEY and the MESSAGE clause define the position of the function key area and of the message area respectively. Both are positioned to the bottom of the panel because they are defined as BOTTOM.
- The command area is not defined in the COMMAND-LINE clause but activated by the global variables MDG_MMR_CX_CMD_LINE and MDG_MMR_CX_CMD_TYPE for all input panels in the environment. If you change the setting of the global variables a command area is only generated if the COMMAND-LINE clause is specified in the FMT-SCREEN member defining the input panel.
- The SEE clause specifies extended help for the input panel. The help is to be generated from the INFOBANK-PANEL member W33000H. For details of how to define help, refer to "Defining Help" on page 68.
- No strings are defined for the various field control characters, so their defaults are used.
- The DECLARE-FIELDS clause defines three input fields and an output field for the panel. The position and the type (unprotected) of the input fields are indicated by the field control characters ??HU in the CONTENTS clause. You need not indicate the position of the output field for the message area by using field control characters in the CONTENTS clause. This is done automatically.
- Alternatively, you can define the fields used by the panel in separate ITEM members and not in the DECLARE-FIELDS clause, when creating user-defined input panels. Fields defined in ITEM members must be specified in the INPUTS or OUTPUTS clause of the FMT-SCREEN member. If you define the fields of a panel in separate ITEM members, contextual help can be generated from the HELP clause of the ITEM member. If valid entries for a field are defined in the INPUT-VALUE clause of an ITEM member they can be selected from within contextual help. For details of how to generate contextual help defined in ITEM members, refer to "Defining Help" on page 68.

- The first two CALLS clauses define the validity check and the processing of entries in the input panel. In the first CALLS clause the keyword COMBIN YYN specifies an input combination, where entries in the first and in the second input field are mandatory but not permitted in the third input field. In the second CALLS clause the keyword COMBIN YNY specifies an input combination, where entries in the first and in the third input field are mandatory but not permitted in the second input field. Only entries in a valid combination will be processed by the SEXEC member EX-TW33000. Otherwise an error message will be output. The overview of the valid input combinations for an input panel is generated from the COMBIN keywords in the CALLS clauses. The length of the string following the COMBIN keyword must correspond with the number of input fields specified for the input panel, in this case, three. Each valid input combination and its corresponding Executive Routine must be defined in a CALLS clause.

  The last four CALLS clauses define the error conditions and their corresponding messages. Please note that the keyword NOCOMBIN, specified in the last CALLS clause can only be used in FMT-SCREEN members of the type INPUT.

- The CONTENTS clause defines the layout of the input panel, using static text and field control characters. Several field control characters can be entered in one line. A field control character is active until another field control character is found in the same line. Please note that even blank lines must be protected using the relevant control characters. The sequence of the fields defined in the DECLARE-FIELDS or in the INPUTS or OUTPUTS clauses must correspond with the sequence of the fields indicated by their corresponding field control characters in the CONTENTS clause of the FMT-SCREEN member. For instance, if you change the sequence of the input fields in the CONTENTS clause, so that *userviews* becomes the second and *entities* the third input field, the sequence of the global variables defining the fields in the DECLARE-FIELDS clause must be changed accordingly, so MDG_INPUT_TW33000_3 becomes the second and MDG_INPUT_TW33000_2 the third entry.

List Panels

Figure 29 shows a typical example of a list panel. To select a subject from the list, enter `S` or a Manager Products line command in the line command area on the left-hand of the panel.

Figure 29 • Example of a List Panel

```
TD61000                      METHODMANAGER              ToolSet SERVICES

===>

            Fetch         Retrieve Stored Lists

----> To fetch a stored KEPT-DATA list enter s in the Line Command Area.

List of stored KEPT-DATA lists
      Name         Type   Date          Time      Blocks/Records   LogID
-----------------------------------------------------------------------

===== TEMP         KEPT   14 JAN 1993   10.32.20  1      5         DJB1


 F1=Help F3=Return F4=Exit F5=Refresh F7=Backward F8=Forward
```

The list panel is defined in the FMT-SCREEN member SC-TD61000 shown in Figure 30.

Figure 30 • Definition of a List Panel

```
FMT-SCREEN
SEE 'D61000H' FOR HELP
TYPE LIST
DECLARE-FIELDS
  'IN MDG_INPUT_TD61000_SELECT ALPHANUMERIC 5'
  'IN MDG_INPUT_TD61000_OUTLST ALPHANUMERIC 76'
  'OUT MDG_MESSAGE_AREA ALPHANUMERIC 79'
FUNCTION-KEY
  'BOTTOM'
MESSAGE
  'BOTTOM'
HEADER
  '         ??HPMETHODMANAGER??XP                ToolSet SERVICES'
OPTION REUSE
CALLS
  EX-TD61000
 AT 'INIT'
CALLS
  EZTD61000S
 AT 'SELECT S'
 PASSING MDG_INPUT_TD61000_OUTLST
CALLS
  EX-TD61000
 AT 'EXIT'
CALLS
  MPEAN0000
 AT 'NOINPUT MESSAGE 43023 I'
CALLS
  MPEAN0000
 AT 'NOTFOUND MESSAGE 43001 I'
CALLS
  MPEAN0000
 AT 'NOTPGM MESSAGE 00110 E'
CONTENTS
??XP
??XP              Fetch         ??HPRetrieve Stored Lists??XP
??XP
??XP ----> To fetch a stored KEPT-DATA list enter??HPs??XPin the Line
Command Area.
??XP
??XP  List of stored KEPT-DATA lists
??XP  Name         Type   Date          Time      Blocks/Records   LogID
??XP  ------------------------------------------------------------------
??BO
??XU_   ??XU
```

The FMT-SCREEN member SC-TD61000 shown in Figure 30 on page 56 defines the list panel as follows:

- The LIST keyword defining the TYPE clause specifies that a list panel is to be generated.
- OPTION REUSE indicates that the panel will be displayed as it was originally, that is showing previous entries made when you return to the list panel from another panel.
- The HEADER clause defines the standard heading for the panel. Please note that field control characters can also be used in the definition.
- The FUNCTION-KEY and the MESSAGE clause define the position of the function key area and of the message area respectively. Both are positioned to the bottom of the panel because they are defined as BOTTOM.
- The command area is not defined in the COMMAND-LINE clause but activated by the global variables MDG_MMR_CX_CMD_LINE and MDG_MMR_CX_CMD_TYPE for all list panels in the environment. If you change the setting of the global variables a command area is only generated if the COMMAND-LINE clause is specified in the FMT-SCREEN member defining the list panel.
- The SEE clause specifies extended help for the input panel. The help is to be generated from the INFOBANK-PANEL member D61000H. For details of how to define help, refer to "Defining Help" on page 68.
- No strings are defined for the various field control characters, so their defaults are used.
- The DECLARE-FIELDS clause defines the input and output fields of the list panel. The position and the type (protected/unprotected) of the fields are indicated by field control characters in the CONTENTS clause. The first input field (MDG_INPUT_TD61000_SELECT) receives the selection character `S` or a Manager Products line command. The position and type of the field (unprotected) is indicated by the control characters ??XU in the last line of the CONTENTS clause. The second field (MDG_INPUT_TD61000_OUTLST) displays the name, type, date, etc. of a stored KEPT-DATA list. MDG_INPUT_TD61000_OUTLST is defined as an input field to enable the user to overwrite the displayed output, for instance by specifying a new name and start a selection again. The position and type (unprotected) of the second input field is indicated by the control characters ??XU in the last line of the CONTENTS clause. To protect the output, define MDG_INPUT_TD61000_OUTLST as output field in the DECLARE-FIELDS clause and change its field control characters in the CONTENTS clause. The output field MDG_MESSAGE_AREA defines the message area. You need not indicate the position of the output field for the message area by using field control characters in the CONTENTS clause. This is done automatically.
- The CALLS clause defines the processing of the list panel. The first clause defines the set up of the panel at initialization time, before the panel displays. The set up is executed by the SEXEC member EX-TD61000. The second CALLS clause defines

the processing of selected KEPT-DATA lists when the user enters the selection character `S` in the line command area and presses Enter. The global variable MDG_INPUT_TD61000_OUTLST that contains the name, type, date, etc. of the selected KEPT-DATA list is passed to the SEXEC member EZTD61000S for further processing. The third CALLS clause defines the processing of the panel when the user presses PF3 or PF4 to finish with this function. Processing is carried out by the SEXEC member EX-TD61000 that either returns the user to the previous panel (PF3) or to the previous application point (PF4). The last three CALLS clauses define the error conditions and their corresponding messages.

- The CONTENTS clause defines the layout of the list panel, using static text and field control characters. Several field control characters can be entered in one line. A field control character is active until another field control character is found in the same line. Please note that even blank lines must be protected using the relevant control characters. The sequence of the fields defined in the DECLARE-FIELDS or in the INPUTS or OUTPUTS clauses must correspond with the sequence of the fields indicated by their corresponding field control characters in the CONTENTS clause of the FMT-SCREEN member. The LIST-BODY characters ??BO indicate that the two input fields of the following line are to be repeated. The number of repetitions is determined by the first input field that follows the LIST-BODY characters (??BO). In this example the first input field is defined in the global variable MDG_INPUT_TD61000_SELECT. Its position is indicated by the control characters ??XU in the CONTENTS clause. The global variable MDG_INPUT_TD61000_SELECT is set to the relevant value in the SEXEC member EX-TD61000 at initialization time of the panel.

Output Panels

Figure 31 shows a typical example of an output panel. It displays information that cannot be processed further from the panel.

Figure 31 • Example of an Output Panel

```
TZ50000                      METHODMANAGER              ToolSet SERVICES

===>

               Information    Display Current User Information

       Logon identifier   :  MAX /  Administrator
       Date               :  06 APR 1992
       Time               :  13.36.26

       Repository user    :  MAX
       Current repository :  ADMIN
       Current status     :  DEV-E

       The above data indicates the user and repository
       currently active.









 F1=Help F3=Return F4=Exit F5=Refresh F7=Backward F8=Forward
```

The output panel is defined in FMT-SCREEN member SC-TZ50000 shown in Figure 32.

Figure 32 • Definition of an Output Panel

```
FMT-SCREEN
TYPE OUTPUT
HEADER
'         ??HPMETHODMANAGER??XP                ToolSet SERVICES'
FUNCTION-KEY 'BOTTOM'
MESSAGE 'BOTTOM'
DECLARE-FIELDS
'OUT MDG_MMR_OUTAREA ALPHANUMERIC 79'
'OUT MDG_MESSAGE_AREA ALPHANUMERIC 79'
SEE Z50000H FOR 'HELP'
OPTION REUSE
CALLS EX-TZ50000 AT 'INIT'
CONTENTS
??XP
??XP              Information   ??HPDisplay Current User Information
??XP
??BO
??XP??FF
```

The FMT-SCREEN member SC-TZ50000 shown in Figure 32 defines the output panel as follows:

- The OUTPUT keyword defining the TYPE clause specifies that an output panel is to be generated.
- OPTION REUSE indicates that the panel will be displayed as it was originally when you return to the output panel from another panel.
- The HEADER clause defines the standard heading for the panel. Please note that field control characters can also be used in the definition.
- The FUNCTION-KEY and the MESSAGE clause define the position of the function key area and of the message area respectively. Both are positioned to the bottom of the panel because they are defined as BOTTOM.
- The command area is not defined in the COMMAND-LINE clause but activated by the global variables MDG_MMR_CX_CMD_LINE and MDG_MMR_CX_CMD_TYPE for all output panels in the environment. If you change the setting of the global variables a command area is only generated if the COMMAND-LINE clause is specified in the FMT-SCREEN member defining the output panel.
- No strings are defined for the various field control characters, so their defaults are used.

- The DECLARE-FIELDS clause defines two output fields. The first field (MDG_MMR_OUTAREA) is set to the current user information. The second field (MDG_MESSAGE_AREA) receives messages. The position and the type (protected) of the first output field is indicated by the field control characters ??XP??FF in the last line of the CONTENTS clause. You need not indicate the position of the output field for the message area by using field control characters in the CONTENTS clause. This is done automatically.
- The SEE clause specifies extended help for the output panel. The help is to be generated from the INFOBANK-PANEL member Z50000H. For details of how to define help, refer to "Defining Help" on page 68.
- The CALLS clause specifies the processing of the output panel. The global variable MDG_MMR_OUTAREA that defines the first output field is set to the current user information in the SEXEC member EX-TZ50000 at initialization time of the panel.
- The CONTENTS clause defines the layout of the output panel, using static text and field control characters. Several field control characters can be entered in one line. A field control character is active until another field control character is found in the same line. Please note that even blank lines must be protected using the relevant control characters. The sequence of the fields defined in the DECLARE-FIELDS or in the INPUTS or OUTPUTS clauses must correspond with the sequence of the fields indicated by their corresponding field control characters in the CONTENTS clause of the FMT-SCREEN member. The LIST-BODY characters ??BO indicate that the output field of the following line is to be repeated. The number of repetitions is determined by the first output field that follows the LIST-BODY characters (??BO). In this example the first output field is defined in the global variable MDG_MMR_OUTAREA. Its position is indicated by the control characters ??XP??FF in the CONTENTS clause. The global variable MDG_MMR_OUTAREA is set to the relevant value in the SEXEC member EX-TZ50000 at initialization time of the panel.

Tailoring Panels of the Update Cycle

The update cycle is a combination of member type cluster menus, list panels, assisted update skeletons, and update help panels.

Member types that are related to each other, for instance all member types that define a repository information model (RIM), are grouped together in member type groups called clusters. Member type groups are selected from member type cluster menus.

A member type group selected from a member type cluster menu displays in a list panel. All member types belonging to a selected member type group are displayed with their naming convention prefixes.

A member selected from a list panel of the update cycle displays in an assisted update skeleton. The skeleton contains the clauses that define the selected member and a short descriptive text for each clause.

Help for an assisted update skeleton is provided in an update help panel. It informs you about the purpose of each clause defining the member. Use PF1 to call an update help panel from an assisted update skeleton.

Member type cluster menus, assisted update skeletons, list and help panels of the update cycle are generated from the repository information model (RIM) that defines their contents.

Refer to "Panel Interface" on page 32 for examples of a member type cluster menu, list panel, assisted update skeleton and a help panel.

Whenever you change the definition of a RIM and enable it, the RIM-dependent components of the panel interface change accordingly. For instance, if you define a new MEMBER-TYPE-GROUP for an existing RIM, the group will automatically be added to the relevant member type cluster menus once the RIM has been successfully enabled.

To change the contents of assisted update skeletons and panels of the update cycle, you need to:

- Change the definition of the relevant member types of the RIM
- Carry out the required generation RULEs

For details of RIM member types, refer to Chapter 9, "Member Types," on page 215.

For details of the enabling process, refer to Chapter 5, "Enabling the Environment," on page 89.

Changes that neither affect the RIM nor require its re-enabling can be made to the layout of assisted update skeletons and panels of the update cycle.

To tailor the layout of member type cluster menus and list panels, change the definition of their FMT-SCREEN members. For details of FMT-SCREEN members defining menus and list panels, refer to Chapter 4, "Defining Panels of Different Types," on page 48.

To tailor the layout of assisted update skeletons, use global variables. For details of global variables for tailoring the assisted update, refer to Chapter 6, "Customizing the Environment," on page 103.

Using Assisted Update on Views

You can use the assisted update facility to update a subset of the clauses contained in a member. This subset of clauses is called a *view* of the relevant member type.

The benefits of defining views of member types are:

Security. You can restrict individual user's access to sensitive data, on a clause-by-clause level.

Ease of use. Users can access just the clauses they need, without searching through a complex member type.

By default, the member type referred to by the view is taken from the UDS-TABLE member that defines your current RIM. You can specify other UDS-TABLE members for member types not specified in your current RIM.

Once defined, the view is available for use by all repositories using that RIM accessed during the MethodManager session.

To create a view on a specific member type in your current environment, you need to *define* it, by:

- Naming the view
- Naming the member type referred to by that view
- Specifying the member type clauses that you wish to view

The view can then be used by specific users. This can be done in two ways:

- On a one-time basis, specifying the view when using the AUPDATE command
- On an ongoing basis, by *selecting* the view via the MMRVIEW command

Views can be selected by users (for example, in a USER-MEMBER), or the Systems Administrator can select views for users (for example, in a LOGON-PROFILE for use when a user logs on).

Once a user has selected a view, the view is active from that point on. Whenever a member of the specified type is updated using assisted update, the view of that member type is given.

You can also:

- Find out information about a defined view
- Delete a defined view, clearing it from the environment

For details on how to use views, refer to the MMR VIEW and the AUPDATE commands, in "MMRVIEW Command" on page 65 and "AUPDATE Command" on page 67.

Example of Using a View

This example assumes you wish to create a view, called RESTRICTGROUP, of the GROUP member type, which contains only these clauses:

- CONTAINS
- ALIAS
- SEE
- NOTE

To define this view, enter:

```
MMRVIEW DEFINE VIEW RESTRICTGROUP FOR GROUP SELECT
             CONTAINS ALIAS SEE NOTE ;
```

You should receive this message:

```
VIEW RESTRICTGROUP FOR GROUP DEFINED
```

Next, each user to which you wish this view to apply must enter this command:

```
MMRVIEW SELECT VIEW RESTRICTGROUP FOR GROUP ;
```

You should receive this message:

```
VIEW RESTRICTGROUP SELECTED FOR GROUP
```

Whenever the relevant users use the assisted update facility to create a GROUP member, they will see the following:

```
*** TOP OF DATA ***
-------------------------------------------------------
* GROUP                                            * GR *
-------------------------------------------------------
* SEE             Reference to other members
??.
-------------------------------------------------------
CONTAINS          Subordinate GROUPS and ITEMS
??.
-------------------------------------------------------
NOTE              One or multi-line text without quotes
??.
-------------------------------------------------------
ALIAS             ALIAS-TYPE and ALIAS-NAME in one line
  ?.  ??.
-------------------------------------------------------
*** END OF DATA ***
```

Whenever the relevant users use the assisted update facility to update an existing GROUP member, they will see all the information that a normal assisted update gives.

However, the clauses shown above displays first; these are updatable. These are followed by the other clauses in a GROUP member type, as with a normal assisted update. These latter clauses are not updatable.

MMRVIEW Command

The MMRVIEW command manipulates a user-defined view of a member type.

Refer to "MMRVIEW Syntax" on page 66 for the syntax of the MMRVIEW command.

Defining a View of a Member Type

To define a view of a specified member type defined in your current RIM, enter:

```
MMRVIEW DEFINE VIEW view-name FOR member-type
                           SELECT clause-list ;
```

where:

view-name is a string defining the name of your view, and can be up to 31 characters long.

member-type is the name of the existing member type of which you wish to take a view. This name is as defined in either:

- The first ENCODE-KEYWORD, for a MEMBER-TYPE member type
- The PRIMARY-NAME clause, for a RELATIONSHIP-TYPE member type

clause-list is a list of identifying keywords of clauses contained in the specified member type. With each keyword you can specify whether that clause is to be updatable (via the UPDATE keyword) or read-only (via the READ keyword). By default, UPDATE is assumed.

To define a view of a member type which is defined in a specific UDS-TABLE member on your MP-AID, enter:

```
MMRVIEW DEFINE VIEW view-name FOR member-type
                  UDS uds-name SELECT clauses ;
```

where *uds-name* is the name of a UDS-TABLE on your MP-AID.

Selecting a View of a Member Type

To select a defined view (making it usable), enter:

```
MMRVIEW SELECT VIEW view-name FOR member-type ;
```

This command can be placed in a user's Logon Profile.

Displaying Information About a View

To display information about a defined view, enter either:

```
MMRVIEW DISPLAY VIEW view-name FOR member-type ;
```

or

```
MMRVIEW DISPLAY VIEW view-name FOR member-type
                                   UDS uds-name ;
```

The information displayed will consist of a list of the clauses available within the member type viewed. Those clauses used by the view are shown as UPDATE. Those not used by the view are shown as READ.

Deleting a View

To delete a defined view of a member type, enter:

```
MMRVIEW DELETE VIEW view-name FOR member-name ;
```

or

```
MMRVIEW DELETE VIEW view-name FOR member-name
                                 UDS uds-name ;
```

Once deleted, the view will no longer be usable: it will have to be re-defined and re-selected to be used with assisted update.

MMRVIEW Syntax

```
>>──MMRVIEW──┬─DEFINE view clause-list──┬──┬─;─┬──>><
             ├─SELECT view──────────────┤  └─.─┘
             ├─DISPLAY view─────────────┤
             └─DELETE view              │
```

where:

view is:

```
>──VIEW view-name FOR member-type──┬──────────────────┬──>
                                   └─UDS uds-name─────┘
```

where:

view-name is a string of up to 31 characters, defining the label of a specific view.

member-type is the name of the existing member type of which you wish to take a view. This name is defined in either:

- The first ENCODE-KEYWORD, for a MEMBER-TYPE member type
- The PRIMARY-NAME clause, for a RELATIONSHIP-TYPE member type

uds-name is the name of a UDS-TABLE member on your MP-AID.

clause-list is:

```
                <<<<<<<<<<<<<<<<<<<<<<<
>------SELECT clause ------------------------------------------------>
                          |---READ---|
                          |---WRITE--|
```

where *clause* is an identifying keyword of a clause contained in the specified member type.

AUPDATE Command

The AUPDATE command updates a repository member.

To update (or create) a repository member, enter:

```
AUPDATE name ;
```

where *name* is the name of a member on the repository.

To specify the type of member, enter:

```
AUPDATE name type ;
```

where *type* is the member type: for example, DB2-TABLE. You must specify the member type in environments with no naming conventions.

To specify whether the member is an entity, relationship, or collective member type, enter:

```
AUPDATE name type option ;
```

where *option* is M (for an entity member type defined by a MEMBER-TYPE), R (for a relationship member type defined by a RELATIONSHIP-TYPE), or C (for a collective member type defined by a COLLECTIVE clause of a HIERARCHY member).

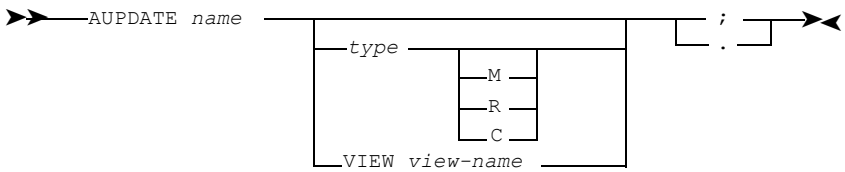
To update a view of a member, enter:

```
AUPDATE name VIEW view-name ;
```

where *view-name* is the name of a previously-defined view.

For details on views of member types, refer to "Using Assisted Update on Views" on page 63.

AUPDATE Syntax



where:

name is the name of a member on the repository.

type is the type of member (for example, DB2-TABLE).

view-name is a string of up to 31 characters, defining the label of a specific view.

Defining Help

You can create two different types of help for the panel interface:

- Extended help that provides information for an entire panel
- Contextual help that provides information for one input field of an input panel

To define extended help:

- Use INFOBANK-PANEL or FMT-SCREEN members if the complete help text is contained in one member
- Use any member type other than INFOBANK-PANEL or FMT-SCREEN if the complete help text is to be generated from several members

To define contextual help for input fields of input panels:

- Use ITEM members that define the fields of a panel
- Use any member type of the Administration Repository if user-defined executive routines generate help from the relevant members

The relationship between a panel and a member defining its help is set up via the SEE clause or the CALLS clause of the FMT-SCREEN member that defines the panel.

Extended help is called using a function key—PF1 by default.

Contextual help is called using the character or string specified in the HELP-IDENTIFIER clause of the FMT-SCREEN member that defines the panel—a question mark (?) by default.

For details of the FMT-SCREEN and ITEM member types, refer to Chapter 9, "Member Types," on page 215.

Defining Extended Help in INFOBANK-PANEL or FMT-SCREEN Members

For the menu in Figure 33 extended help is defined in an INFOBANK-PANEL member.

Figure 33 • Example of a Menu

```
K00000                          METHODMANAGER           ToolSet SERVICES

===>

               Document       Documentation functions

 select one of the following

           1  List           List DOCUMENT members
           2  Project        Select project related DOCUMENT members
           3  Print          Output documents
           4  Structure      Display dependencies of documents
           5  Update         Create and update DOCUMENT members

           U  User           User defined option
           T  Tutorial       Documentation tutorial









  F1=Help F3=Return F4=Exit
```

The menu in Figure 33 on page 69 is defined in the FMT-SCREEN member SC-K00000. Figure 34 shows an excerpt of the member definition.

Figure 34 • Extended Help Specified via the SEE Clause of an FMT-SCREEN Member

```
FMT-SCREEN
SEE K00000H FOR 'HELP'
TYPE MENU
DECLARE-FIELDS
 'IN MDG_MENU_SELECTION1 ALPHANUMERIC 1'
 'OUT MDG_MESSAGE_AREA ALPHANUMERIC 70'
FUNCTION-KEY 'BOTTOM'
```

The INFOBANK-PANEL member K00000H specified in the SEE clause in Figure 34 contains the actual help text for the menu. Figure 35 on page 70 shows an excerpt of the INFOBANK-PANEL member K00000H.

Figure 35 • Extended Help Defined in an INFOBANK-PANEL Member

```
===== HIGHLIGHT
===== £
===== END-OF-FIELD
===== %
===== CONTENTS
=====       Document      £Documentation functions%
=====
=====     The functions listed on this menu enable you to create manuals
=====     and other documents using structured repository information.
=====
=====     The DOCUMENT member type is used for the creation of
=====     documentation. A DOCUMENT member can contain commands as well
=====     as free-form text. A document is created from one or more
=====     DOCUMENT members by processing commands and amalgamating the
=====     results with any text contained in the member(s). Headings and
=====     their numbering are automatically controlled.
```

Figure 36 shows extended help, called using PF1 from the menu in Figure 33 on page 69

Figure 36 • Extended Help Generated from an INFOBANK-PANEL Member

```
HELP                            METHODMANAGER                   K00000H

====> SELECT

            Document       Documentation functions

          The functions listed on this menu enable you to create manuals
          and other documents using structured repository information.

          The DOCUMENT member type is used for the creation of
          documentation. A DOCUMENT member can contain commands as well
          as free-form text. A document is created from one or more
          DOCUMENT members by processing commands and amalgamating the
          results with any text contained in the member(s). Headings and
          their numbering are automatically controlled.

          Option 1 lists DOCUMENT members for processing using Line
                   Commands.

          Option 2 lists DOCUMENT members which have been assigned to a
                   particular project.

_____________________________________________________________ MORE>>>>
F1=Help F3=Return F4=Exit F5=Refresh F7=Backward F8=Forward
```

Defining Extended Help in Any Member Type Other Than INFOBANK-PANEL or FMT-SCREEN

For the menu in Figure 37 extended help can be defined in several members.

Figure 37 • Example of a Menu

```
A00000                        METHODMANAGER           ToolSet SERVICES

===>

               Admin          System administration

 select one of the following

            1  Member Type    Select members by member type
            2  Member         Create and update members
            3  Enable         Enable ToolSet SERVICES

            U  User Defined   User defined option
            T  Tutorial       Administration tutorial









  F1=Help F3=Return F4=Exit
```

The menu in Figure 37 on page 72 is defined in an FMT-SCREEN member. The definition in Figure 38 is a tailored version of the FMT-SCREEN member SC-A00000. It shows as an example how extended help can be generated from several ITEM members.

Figure 38 • Extended Help Specified via the SEE Clause of an FMT-SCREEN Member

```
===== *** TOP OF DATA ***
===== FMT-SCREEN
===== CATALOG
=====  'MMR'
===== ,'GR2'
===== ,'MR0'
===== ,'TSS'
===== ,'EXTERNAL'
===== SEE MDG_A00000_1 FOR 'HELP'
=====    ,MDG_A00000_2 FOR 'HELP'
=====    ,MDG_A00000_3 FOR 'HELP'
=====    ,MDG_A00000_U FOR 'HELP'
=====    ,MDG_A00000_T FOR 'HELP'
===== TYPE MENU
===== DECLARE-FIELDS
=====   'IN MDG_MENU_SELECTION1 ALPHANUMERIC 1'
=====   'OUT MDG_MESSAGE_AREA ALPHANUMERIC 70'
===== FUNCTION-KEY
```

The five ITEM members specified in the SEE clause in Figure 38 on page 73 define a title and a short help text for each option of the menu. Figure 39 and Figure 40 show the definition of the first two ITEM members MDG_A00000_1 and MDG_A00000_2.

Figure 39 • Example of an ITEM Member Defining the First Part of an Extended Help Text

```
===== *** TOP OF DATA ***
===== ITEM
===== CATALOGUE
===== 'MMR'
===== ,'GR2'
===== ,'MR0'
===== ,'TSS'
===== ,'EXTERNAL'
===== NOTE 'THIS ITEM FORMS PART OF THE ONLINE HELP FOR SC-A00000'
===== TITLE
===== 'Select members by member type'
===== HELP
===== Use this option to select members of the Administration
===== Repository.
===== *** END OF DATA ***
```

Figure 40 • ITEM Member Defining the Second Part of an Extended Help Text

```
===== *** TOP OF DATA ***
===== ITEM
===== CATALOGUE
===== 'MMR'
===== ,'GR2'
===== ,'MR0'
===== ,'TSS'
===== ,'EXTERNAL'
===== NOTE 'THIS ITEM FORMS PART OF THE ONLINE HELP FOR SC-A00000'
===== TITLE
===== 'Create and update members'
===== HELP
===== Use this option to create and update members which, for
===== instance define a RIM or the user interface in the
===== Administration Repository.
===== *** END OF DATA ***
```

From the ITEM members specified in the SEE clause in Figure 38 on page 73 one complete extended help is generated when the FMT-SCREEN member is constructed onto the MP-AID. Figure 41 shows the generated help, called using PF1 from the menu in Figure 37 on page 72.

Figure 41 • Extended Help Generated from Several ITEM Members

```
TH13000              METHODMANAGER                    ToolSet SERVICES

                                Select members by member type

 Use this option to select members of the Administration
 Repository.

                                Create and update members

 Use this option to create and update members which, for
 instance define a RIM or the user interface in the
 Administration Repository.

                                Enable ToolSet SERVICES

 Use this option to activate the user interface for a defined
 Repository Information Model (RIM).

                                User defined option



 F1=Help F3=Return F4=Exit F5=Refresh F9=to CMR
```

Defining Contextual Help in ITEM Members

Contextual help can be defined for input fields of an input panel. It is specified via the SEE clause of the FMT-SCREEN member defining the input panel.

For the input panel in Figure 42 contextual help is defined in nine ITEM members. Additionally extended help is defined in an INFOBANK-PANEL member.

The input panel in Figure 42 is defined in the FMT-SCREEN member SC-TD34000. Figure 43 shows an excerpt from the member definition.

Figure 42 • Example of an Input Panel

```
TD34000                        METHODMANAGER          ToolSet SERVICES

===>

               Text           Search for Attributes Containing Text

process existing KEPT-DATA list ----> ______________________________
member type(s)                  ----> ____________________________________
ER integrity indicator          ----> _____________
with the attribute              ----> ___________________________________
with the string                 ----> ____________________________________
from line                       ----> ___
to line                         ----> ___
exclude member type(s)          ----> ____________________________________
keep results in KEPT-DATA list  ----> _________________________________

  Use this panel to search for members that contain a specific
  attribute or that contain a specific text string within that
  attribute.
```

Figure 43 • Contextual Help Specified via the SEE Clause of an FMT-SCREEN Member

```
SEE D34000H FOR 'HELP'
,MDG_TD34000_KEPT1 FOR 'HELP-ID MDG_TD34000_KEPT1'
,MDG_TD34000_MT FOR 'HELP-ID MDG_TD34000_MT'
,MDG_TD34000_ER FOR 'HELP-ID MDG_TD34000_ER'
,MDG_TD34000_ATR FOR 'HELP-ID MDG_TD34000_ATR'
,MDG_TD34000_STR FOR 'HELP-ID MDG_TD34000_STR'
,MDG_TD34000_FL FOR 'HELP-ID MDG_TD34000_FL'
,MDG_TD34000_TL FOR 'HELP-ID MDG_TD34000_TL'
,MDG_TD34000_EX FOR 'HELP-ID MDG_TD34000_EX'
,MDG_TD34000_KEPT2 FOR 'HELP-ID MDG_TD34000_KEPT2'
INPUTS
 MDG_TD34000_KEPT1
,MDG_TD34000_MT
,MDG_TD34000_ER
,MDG_TD34000_ATR
,MDG_TD34000_STR
,MDG_TD34000_FL
,MDG_TD34000_TL
,MDG_TD34000_EX
,MDG_TD34000_KEPT2
OUTPUTS
 MDG_MESSAGE_AREA
```

The member D34000H defines extended help, specified first via the SEE clause in Figure 43 on page 76. "Defining Extended Help in INFOBANK-PANEL or FMT-SCREEN Members" on page 69 describes how to set up extended help from INFOBANK-PANEL members. The other members which start with the prefix MDG_are ITEM members, defining the input fields of the panel. The contextual help is defined in the HELP clause of the ITEM member. Figure 44, Figure 45, and Figure 46 on page 78 show the definitions of the first three ITEM members (MDG_TD34000_KEPT1, MDG_TD34000_MT, and MDG_TD34000_ER).

Figure 44 • Example of an ITEM Member Defining the First Input Field of the Panel in Figure 42 on page 76

```
ITEM
DEFAULTED-AS ALPHANUMERIC 32
TITLE
'process existing KEPT-DATA list'
MODE UPPER
HELP
     To interrogate only those members held in an existing KEPT-DATA
     list, enter the name of that KEPT-DATA list.
```

Figure 45 • Example of an ITEM Member Defining the Second Input Field of the Panel in Figure 42 on page 76

```
ITEM
DEFAULTED-AS ALPHANUMERIC 43
TITLE
'member type(s)'
MODE UPPER
HELP
     To interrogate members of a specific member type only, enter
     the name(s) of those member types. If you enter more than one
     member type, each must be separated by a comma (,).
```

Figure 46 • Example of an ITEM Member Defining the Third Input Field of the Panel in Figure 42 on page 76

```
ITEM
DEFAULTED-AS ALPHANUMERIC 12
TITLE
'ER integrity indicator'
MODE UPPER
INPUT-VALUE
 'CHECK-OK; select members already checked'
,'CHECK-NEEDED; select members to be checked'
HELP
     Members can be selected via their ER integrity indicator.
     To select members already checked, select CHECK-OK.
     To select members to be checked, select CHECK-NEEDED.
     To select both, enter nothing.
```

By default the contextual help is generated from the TITLE and the HELP clause, which define respectively the title and the help text for the input field.

You can change these defaults for each member type, using the global variable MDG_MMR_CX_HELP_TYPE. To change the default, specify the names of the relevant clauses in MDG_MMR_CX_HELP_TYPE.

Note:

Input values can be defined for an ITEM member, giving a list of valid values which displays with the help text. Once selected in the contextual help, the value is automatically inserted into the input field of the input panel. In Figure 46 the input values CHECK-OK and CHECK-NEEDED have been defined in the ITEM member.

Figure 47 shows the contextual help generated from the ITEM member in Figure 46. To call the help, enter the character or the string defined in the HELP-IDENTIFIER clause of the FMT-SCREEN member in the third field of the input panel shown in Figure 42 on page 76.

Figure 47 • Contextual Help Generated from an ITEM Member

```
TH10000              METHODMANAGER                    ToolSet SERVICES

===>

 ER integrity indicator

         Members can be selected via their ER integrity indicator.
         To select members already checked, select CHECK-OK.
         To select members to be checked, select CHECK-NEEDED.
         To select both, enter nothing.
S  CHECK-OK      select members already checked
_  CHECK-NEEDED  select members to be checked









 F1=Help F3=Return F4=Exit F5=Refresh
```

If the input value CHECK-OK is selected, this value is automatically inserted into the input field in the input panel, as shown in Figure 48.

Figure 48 • Example of an Input Panel Displaying a Value Inserted Automatically from within Contextual Help

```
TD34000                        METHODMANAGER           ToolSet SERVICES

===>

               Text           Search for Attributes Containing Text

 process existing KEPT-DATA list ----> ________________________________
 member type(s)                  ----> ___________________________________
 ER integrity indicator          ----> CHECK-OK
 with the attribute              ----> ________________________________
 with the string                 ----> ___________________________________
 from line                       ----> ___
 to line                         ----> ___
 exclude member type(s)          ----> ___________________________________
 keep results in KEPT-DATA list  ----> ________________________________

  Use this panel to search for members that contain a specific
  attribute or that contain a specific text string within that
  attribute.



 F1=Help F3=Return F4=Exit F5=Refresh
```

Defining Contextual Help in Any Member Type of the Repository

Contextual help specified via the CALLS clause of a FMT-SCREEN member can be defined in any member type of the repository. In this case the generation of help for a field needs to be executed by a user-defined executive routine.

Figure 49 is an example that shows the definition of an input panel in a FMT-SCREEN member.

Figure 49 • Contextual Help Specified via the CALLS Clause of an FMT-SCREEN Member

```
FMT-SCREEN
TYPE INPUT
OPTION HOLD
HEADER
'METHODMANAGER                          ToolSet SERVICES'
FUNCTION-KEY 'BOTTOM'
MESSAGE 'BOTTOM'
APPLICATION-POINT YES
CALLS EN4711 AT 'INIT'
CALLS EN4711 AT 'PROCESS'
CALLS EN4711 AT 'EXIT'
CALLS EN4711 AT 'CANCEL'
CALLS EH6610 AT 'HELP' PASSING MDG_UTR2RU_ATTR_1
CALLS EH6610 AT 'HELP' PASSING MDG_UTR2RU_ATTR_2
CALLS EH6610 AT 'HELP' PASSING MDG_UTR2RU_ATTR_3
```

Members starting with the prefix MDG_ are ASG-supplied global variables defined in ITEM members. They specify the fields for which contextual help is to be generated.

Members starting with the prefix EH are SEXEC members. These SEXEC members do *not* contain the actual help text but define the generation of help. For instance, the SEXEC member might first check a condition and then, depending on the result, call the appropriate member that contains the help text. If contextual help is defined in a FMT-SCREEN member use the :FMTSCREEN macro to call that member.

For details of the :FMTSCREEN macro, refer to "Macros" on page 195.

Checking the Layout of a Defined Panel

To check the layout, fields and valid input combinations of a defined panel before it is constructed onto the MP-AID, enter:

```
FMTOUT name ;
```

name is the repository name of a FMT-SCREEN member. The FMT-SCREEN member must be successfully encoded before the FMTOUT command can be applied to it.

In the following the output of the FMTOUT command is described in three parts.

In Figure 50, the following apply:

- Use the scale at the top of the panel to check the position of static text, input, and output fields.
- Underscores (_) represent the position of input fields in the panel, asterisks (*) the position of output fields. Asterisks (*) also indicate the position of the message area and of the function key area.

Note:

The output displayed in Figure 50, Figure 51, and Figure 52 has been truncated on the right.

First the layout of the defined panel is shown in the output.

Figure 50 • Output of the FMTOUT Command - First Part

```
*** TOP OF DATA ***

LAYOUT OF THE DEFINED SCREEN

UNDERSCORES (_) REPRESENT INPUT  FIELDS
ASTERISKS   (*) REPRESENT OUTPUT FIELDS

0        1         2         3         4         5         6         7
1---5----0----5----0----5----0----5----0----5----0----5----0----5----0-
.......................................................................
. TD33000                        METHODMANAGER                 ToolSe
.
. ===>
.
.               Value          Search for Attributes Containing Values
.
. process existing KEPT-DATA list ----> ______________________________
. member type(s)                  ----> ______________________________
. ER integrity indicator          ----> ___________
. with the attribute              ----> ______________________________
. and the value                   ----> ___________________________________     .
. keep results in KEPT-DATA list  ----> ___________________________________     .
.
.  Use this panel to search for members that contain a specific
.  attribute or that contain a specific value within that attribute.
.
.
.
.
.
.
.
.
.
. ************************************************************************
.
. ************************************************************************

.......................................................................
```

FMTOUT now lists all fields used by the FMT-SCREEN member defining the panel, showing the exact positions of static text, input and output fields.

Figure 51 • Output of FMTOUT Command - Second Part

```
=====================================================================
FIELDS USED BY THE DEFINED SCREEN
INPUT  MDG_COMMAND_LINE
INPUT  MDG_INPUT_TD33000_1
INPUT  MDG_INPUT_TD33000_2
INPUT  MDG_INPUT_TD33000_6
INPUT  MDG_INPUT_TD33000_3
INPUT  MDG_INPUT_TD33000_4
INPUT  MDG_INPUT_TD33000_5
OUTPUT MDG_MESSAGE_AREA
OUTPUT MDG_MMR_FUNCTIONTEXT

POSITION OF STATIC STRINGS, INPUT AND OUTPUT FIELDS ON THE SCREEN
LINE COL LEN OCC ATT ID FIELD/STRING
 01   02  20      XP  S 'TD33000             '
 01   34  13      HP  S 'METHODMANAGER'
 01   48  32      XP  S '                ToolSet SERVICES'
 02   02  01      XP  S ' '
 03   02  04      HP  S '===>'
 03   07  72      XU  S MDG_COMMAND_LINE
 04   02  01      XP  S ' '
 05   02  28      XP  S '              Value         '
 05   31  39      HP  S 'Search for Attributes Containing Values'
 06   02  02      XP  S '  '
 07   02  37      XP  S 'process existing KEPT-DATA list ---->'
 07   40  40      HU  I MDG_INPUT_TD33000_1
 08   02  37      XP  S 'member type(s)                  ---->'
 08   40  40      HU  I MDG_INPUT_TD33000_2
 09   02  37      XP  S 'ER integrity indicator          ---->'
 09   40  12      HU  I MDG_INPUT_TD33000_6
 10   02  37      XP  S 'with the attribute              ---->'
 10   40  40      HU  I MDG_INPUT_TD33000_3
 11   02  37      XP  S 'and the value                   ---->'
 11   40  40      HU  I MDG_INPUT_TD33000_4
 12   02  37      XP  S 'keep results in KEPT-DATA list  ---->'
 12   40  40      HU  I MDG_INPUT_TD33000_5
 13   02  02      XP  S '  '
 14   02  62      XP  S ' Use this panel to search for members that
 15   02  66      XP  S ' attribute or that contain a specific value
 16   02  01      XP  S ' '
 17   02  01      XP  S ' '
 18   02  01      XP  S ' '
 19   02  01      XP  S ' '
 20   02  79      HP  O MDG_MESSAGE_AREA
 21   02  79      XP  O MDG_MMR_FUNCTIONTEXT
```

The repository names of all fields used by the panel are listed in the sequence defined in the DECLARE-FIELDS or INPUTS/OUTPUTS clause of the FMT-SCREEN member.

Field specific information displays in a table below the listed fields in Figure 51. The table is to be read from left to right.

Explanation of the table headings:

LINE. Displays the line number of a static string or a field in the panel.

COL. Displays the column number that indicates the beginning of a static string or a field in the panel.

LEN. Displays the number of characters a static string or a field consists of.

OCC. Displays a number that indicates the repetition of fields in the actual output. A number displays only when fields are to be repeated in a FMT-SCREEN member of the type LIST.

ATT. Displays a field control character that indicates whether a static string or a field is protected/unprotected or output highlighted or with normal intensity.

ID. Displays:

- S = indicator of a static string
- I = indicator of an input field
- O = indicator of an output field

FIELD/STRING. Displays the repository name of an input/output field or a static string.

Finally the FMTOUT command displays the valid input combinations of an input panel.

Figure 52 • Output of FMTOUT Command - Third Part

```
VALID INPUT COMBINATIONS OF THE DEFINED SCREEN

 0        1         2         3         4         5         6         7
 1---5----0----5----0----5----0----5----0----5----0----5----0----5----0-
.......................................................................
. TD33000                        METHODMANAGER                 ToolSet
.
. ===>
.
.               Value          Search for Attributes Containing Values
.
. process existing KEPT-DATA list ----> +********
. member type(s)                  ----> -++++----
. ER integrity indicator          ----> *********
. with the attribute              ----> -++++++++
. and the value                   ----> ---++--++
. keep results in KEPT-DATA list  ----> --+-+-+-+
.
.  Use this panel to search for members that contain a specific
.  attribute or that contain a specific value within that attribute.
.
.
.
.  ----------------------------------------------------------
.  + Mandatory Input,   * Optional Input,   - No Input Allowed
.  ----------------------------------------------------------
. Each column of the above symbol table describes a valid
. combination of inputs. To read the screen, look down each
. column, matching up each symbol with its corresponding input.
.
. ***********************************************************************
. ***********************************************************************
.......................................................................
*** END OF DATA ***
```

The output in Figure 52 will only be generated if the FMTOUT command is applied to a FMT-SCREEN member of the type INPUT and if the COMBIN keyword is specified in the CALLS clause.

Enabling a Defined Panel

To enable a panel defined in a FMT-SCREEN member, use the CX command.

The CX command constructs repository members onto the MP-AID.

CX Command Syntax

```
>>──CX member-name ──┬─;─┬──────────────────────────────────><
                     └─.─┘
```

where *member-name* is the name of one of these members:

- EXECUTIVE-ROUTINE
- FMT-SCREEN
- FORMAT
- GLOBAL-PROFILE
- INFOBANK-PANEL
- LOGON-PROFILE
- SEXEC

5 Enabling the Environment

This chapter includes these sections:

How to Enable Your Environment 90
Enabling HDS Tables 92

Complete Generation 98

Partial Generation. 99

Analyzing Generated Executives 99

How to Disable an Environment 101

The UX COMMAND 101
UX Command Syntax 101

Having defined the repository information model (RIM) in the Administration Repository, you need to enable the RIM to set up an interactive environment.

Enabling involves:

- Generating the RIM-dependent components of the user interface, for instance cluster menus, list panels or the update help
- Activating the ASG-supplied components of the user interface for the defined RIM
- Generating a UDS-TABLE member from the RIM definition
- Assigning the enabled environment to an existing corporate repository.

You may need to completely or partially generate the RIM, depending on the changes you have made. For instance, if you have tailored an ASG-supplied RIM, partial generation might be sufficient. Refer to "Table 5.1a Member Type: HIERARCHY" on page 92 for details of the required action for individual changes.

How to Enable Your Environment

You can enable the environment via the panel interface (menu A70000) or using the UX, CONSTRUCT, CONTROL, and COMPARE commands.

Generation is carried out in several steps, each step represented by a separate RULE. A RULE executes specific actions in the enabling process, necessary to set up an interactive environment for a corporate repository.

ASG recommends executing RULE010, RULE020, RULE030, RULE100, RULE120, and RULE130 interactively via the panel interface. RULE040 to RULE080 should be done in batch using the UX command.

Note:

Before you enable your environment via the panel interface, you must select a HIERARCHY member to identify the RIM to be generated.

The RULEs and their Functions:

RULE010: Check ALIAS 1

This RULE checks that a unique two-character entry for the ALIAS clause of the MEMBER-TYPE members and RELATIONSHIP-TYPE members has been specified

RULE020: Check ENCODE-KEYWORDS

This RULE checks that the ENCODE-KEYWORDS or the LONG-NAME clause of the MEMBER-TYPE members has been defined

RULE030: Check Attribute Types

This optional rule checks whether any DUMMY members have been specified in the SEE clauses of MEMBER-TYPE and RELATIONSHIP-TYPE members and whether all ATTRIBUTE-TYPE members in the ATTRIBUTES clause and all appropriate GENERIC ATTRIBUTE-TYPES are also specified in the SEE clause. In addition it checks whether there are any COMMON ATTRIBUTES in the HIERARCHY which are not used by any MEMBER/RELATIONSHIP-TYPE definition.

RULE040: Enable Update Help

This RULE generates the update help for the assisted update skeletons. The help is generated from the HELP clauses of the MEMBER-TYPE, RELATIONSHIP-TYPE, and ATTRIBUTE-TYPE members.

RULE050: Enable Cluster Menus and List Panels

This RULE generates the member type cluster menus and list panels. They are generated from the CONTAINS and RELATIONSHIPS clauses of the HIERARCHY member and from the SEE clauses of the MEMBER-TYPE-GROUP members

RULE070: Enable Naming Convention Tables

This RULE generates naming convention tables and check routines for the naming conventions of members from the NAMING clause of the MEMBER-TYPE and RELATIONSHIP-TYPE members

RULE080: Enable Update Skeletons

This RULE generates the assisted update skeletons for entity and relationship member types.

RULE100: Construct RIM

This RULE validates the RIM HIERARCHY member and constructs from it a UDS-TABLE member on the MP-AID in the same manner as a CONSTRUCT UDS-TABLE command.

RULE110: Compare RIM

This RULE compares two UDS-TABLEs in order to assess their compatibility in the same manner as a COMPARE UDS-TABLE command.

RULE120: Control RIM in Repository

This RULE implements a UDS-TABLE in a particular repository in the same manner as a CONTROL UDS command.

RULE130: Control UDR in Repository

This RULE renames UDR clauses in the same manner as a CONTROL UDR command. The clauses must be specified in the SEE clause of the HIERARCHY member.

For details of the CONSTRUCT UDS-TABLE, COMPARE UDS-TABLE, CONTROL UDS and CONTROL UDR commands, refer to *ASG-Manager Products Controller's Manaul*.

Enabling HDS Tables

A HDS table is the programmable workstation (PGW) representation of a UDS-TABLE member on the MP-AID. A HDS table defines the syntax of local repository members and enables you to create and maintain corporate repository information on the PGW.

The generation of a HDS table requires that the mainframe environment of a corporate repository has been successfully enabled.

A HDS table and an associated Translation Executive Routine is generated from the definition of a HDS-TABLE member in the Administration Repository.

Once you have enabled a HDS table, the table must be downloaded to the PGW.

For details of HDS Table and the MVW-GEN command, refer to *ASG-ManagerView System Administrator's Tailoring*.

Table 5.1 details the RULEs that have to be started in order to enable the changes you have made to a RIM.

Table 5.1a Member Type: HIERARCHY

New or Changed Definition	RULES to be Started											
	ALL	010	020	030*	040	050	070	080	100	110	120	130
ALTERNATIVE-RELATIONSHIP									●	●	●	
COLLECTIVE							●		●	●	●	
COMMON-ATTRIBUTES									●	●	●	
CONTAINS	●											
MPAID-NAME	●											
RELATIONSHIPS		●			●	●	●	●	●	●	●	
RELATIONSHIPS-VALUE									●	●	●	
SEE								●				●
SYNONYM									●	●	●	
UDO									●	●	●	

*RULE30 is optional

Table 5.1b Member Type: MEMBER-TYPE-GROUP

New or Changed Definition	RULES to be Started											
	ALL	010	020	030*	040	050	070	080	100	110	120	130
CONTAINS	●											
OPTION						●						
OPTION-NAME						●						
OPTION-TEXT						●						
SEE		●			●	●	●	●				

*RULE030 is optional

Table 5.1c Member Type: MEMBER-TYPE

New or Changed Definition	RULES to be Started											
	ALL	010	020	030*	040	050	070	080	100	110	120	130
ALIAS		●			●		●	●				
ATTRIBUTES									●	●	●	
AUTO-REF-STRING								●				
BASED-ON									●	●	●	
ENCODE-KEYWORDS			●		●	●	●	●	●	●		
GENERIC-ATTRIBUTES									●	●	●	
HELP					●							
INTERROGATE-KEYWORDS							●		●	●	●	
IS									●	●	●	
LEVEL									●	●	●	
LONG-LITERAL									●	●	●	
LONG-NAME					●	●	●	●				
NAMING						●	●					

Table 5.1c Member Type: MEMBER-TYPE

New or Changed Definition	RULES to be Started										
	ALL	010	020	030*	040	050	070	080	100	110	120	130
NAMING-EXIT							●					
PLURAL-LITERAL							●		●	●	●	
RECURSIVE									●	●	●	
RELATIONSHIPS VIA									●	●	●	
REPORT-DOWN-TO-KEYWORDS									●	●	●	
SEE		●		●	●			●				
SHORT-LITERAL							●		●	●	●	
STANDARD-LITERAL			●		●	●	●	●	●	●	●	

*RULE030 is optional

Table 5.1d Member Type: ATTRIBUTE-GROUP

New or Changed Definition	RULES to be Started										
	ALL	010	020	030*	040	050	070	080	100	110	120	130
CONTAINS					●			●	●	●	●	

*RULE030 is optional

Table 5.1e Member Type: ATTRIBUTE-TYPE

New or Changed Definition	RULES to be Started										
	ALL	010	020	030*	040	050	070	080	100	110	120	130
CHARACTER-STRING								●	●	●	●	
DECIMAL-NUMBER								●	●	●	●	
DATE								●	●	●	●	
EDIT-CODE-1								●				

Table 5.1e Member Type: ATTRIBUTE-TYPE					
EDIT-CODE-2		●			
EDIT-EXEC-1		●			
EDIT-EXEC-2		●			
FREE-FORM-TEXT		●	●	●	●
HELP	●				
IDENTIFIED-BY	●	●	●	●	●
INDEXED-BY			●	●	●
INTEGER		●	●	●	●
KEYWORD		●	●	●	●
LONG-NAME	●	●			
MAXIMUM-LENGTH		●	●	●	●
MAXIMUM-LINES		●	●	●	●
MAXIMUM-NUMBER			●	●	●
MAXIMUM-VALUE			●	●	●
MINIMUM-LENGTH			●	●	●
MINIMUM-LINES			●	●	●
MINIMUM-NUMBER			●	●	●
MINIMUM-VALUE			●	●	●
MULTIPLE- VALUES			●	●	●

Table 5.1e cont. Member Type: ATTRIBUTE-TYPE

New or Changed Definition	RULES to be Started										
	ALL	010	020	030*	040	050	070	080	100	110	120	130
NAME								●	●	●	●	
NAMED								●	●	●	●	
NORMALIZED-MAXIMUM-VALUE									●	●	●	
NORMALIZED-MINIMUM-VALUE									●	●	●	
NORMALIZED-VALUE									●	●	●	
PROMPT-CODE								●				
REPEAT-CODE								●				
SEE								●				
SKELETON-CODE								●				
SKELETON-HELP								●				
SKELETON-TEXT								●				
TEXT								●	●	●	●	
TIME								●	●	●	●	
VALUES									●	●	●	

*RULE030 is optional

Table 5.1f Member Type: RELATIONSHIP-GROUP

New or Changed Definition	RULES to be Started										
	ALL	010	020	030*	040	050	070	080	100	110	120	130
ALTERNATIVE-RELATIONSHIPS								●	●	●		
RELATIONSHIPS					●	●	●	●	●	●	●	

*RULE030 is optional

Table 5.1g Member Type: RELATIONSHIP-TYPE

New or Changed Definition	RULES to be Started										
	ALL	010	020	030*	040	050	070	080	100	110	120	130
ALIAS		●			●		●	●				
ATTRIBUTES									●	●	●	
AUTO-REF-STRING								●				
CLASS									●	●	●	
DUPLICATES									●	●	●	
HELP								●				
INVERSE-NAME							●		●	●	●	
LONG-NAME					●	●	●	●				
NAMING						●	●					
NAMING-EXIT							●					
PLURAL-LITERAL									●	●	●	
PRIMARY-NAME					●	●	●	●	●	●	●	
RECURSION									●	●	●	
RELATIONSHIPS VIA									●	●	●	
SEE				●	●			●				
SHORT-LITERAL									●	●	●	
SOURCE									●	●	●	
STANDARD- LITERAL			●		●	●	●	●	●	●	●	
TARGET									●	●	●	

*RULE030 is optional

Table 5.1h Member Type: RELATIONSHIP-CLASS												
New or Changed Definition	RULES to be Started											
	ALL	010	020	030*	040	050	070	080	100	110	120	130
INVERSE-NAME									●	●	●	
PRIMARY-NAME									●	●	●	
*RULE030 is optional												

Table 5.1i Member Type: HDS-TABLE												
New or Changed Definition	RULES to be Started											
	ALL	010	020	030*	040	050	070	080	100	110	120	130
HDS-TABLE (Note 1)												
Note 1: Use the administration functions to enable HDS Tables												
Note 2: *RULE030 is optional												

Complete Generation

Complete generation must be carried out if a new RIM has been defined from scratch or if an existing RIM has been changed fundamentally. For instance, if you have tailored an existing RIM by defining new MEMBER-TYPEs which are specified in the CONTAINS clause of a MEMBER-TYPE-GROUP member, all RULEs have to be executed.

The sequence of the RULEs is important.

RULE010 and RULE020 can be invoked in any order and must be invoked before any of RULES 040-080. This is because RULE010 and RULE020 check whether mandatory entries for the user interface have been defined and whether or not they are unique in the RIM. RULES 040-080 cannot be generated until both of these rules have been generated successfully.

RULE030 can be invoked at any time and, as it is optional, does not have to be invoked at all.

RULE040, RULE050, RULE070, and RULE080 can be invoked in any order after RULE010, RULE020, and RULE030 (if invoked) have been generated successfully.

RULE100, RULE110, RULE120, and RULE130 must be invoked consecutively. If no UDR clauses have been specified, RULE130 can be omitted.

In panels TA72000 and TA73000 a successfully executed RULE is indicated by the character F and a stamp showing the date and time of generation. A RULE which has not been started yet is indicated by the character O before the name of the RULE.

Partial Generation

Not every change in the definition of the RIM makes complete generation necessary. For instance, if you have made a new entry in the NAMING clause of a MEMBER-TYPE member, only RULE050 and RULE070 must be started to activate the change.

Refer to "Table 5.1a Member Type: HIERARCHY" on page 92 for details of which RULEs have to be started to activate individual changes.

Analyzing Generated Executives

As a result of the enabling process, EXECUTIVE members are generated by RULE050, RULE070 and RULE080 from the definition of a RIM. EXECUTIVEs are constructed onto the MP-AID and execute the run control of the user interface. They must not be changed, as this might cause unpredictable results.

However, the administrator or ASG support personnel may want to analyze EXECUTIVE members to trace errors in the definition of a RIM.

To display an EXECUTIVE member on the MP-AID, use the MP-AID PRINT command. For details of this command, refer to the *ASG-Manager Products Systems Administrator's Manual*.

Table 5.2 gives an overview of the naming conventions of EXECUTIVEs on the MP-AID and indicates the corresponding RULE by which they have been generated.

5.2 Naming Conventions of Generated EXECUTIVE Members on the

Naming Conventions of EXECUTIVE members	Generated by
EA00*aaaaa*	RULE070
EA01*aaaaa*	RULE070
EA02*aaaaa*	RULE070
EA03*aaaaa*	RULE070
EA11*aaaaa*	RULE070
EA12*aaaaa*	RULE070
EA2*aaaaabb*	RULE070
EA4*aaaaa*	RULE050
EA5*aaaaa*	RULE050
EA6*aaaaa*	RULE050
EA7*aaaaa*	RULE050
EA8*aaaaabb*	RULE080
EA9*aaaaabb*	RULE080

MP-AID

where:

aaaaa is the name specified in the MPAID-NAME clause of the HIERARCHY member defining the RIM, or if the member does not contain an MPAID-NAME clause, the first five characters of the HIERARCHY member name.

bb is the suffix, defined in the ALIAS clause of the MEMBER-TYPE or RELATIONSHIP-TYPE member(s).

How to Disable an Environment

To disable the panel interface for an existing RIM, you should delete its generated EXECUTIVE and INFOBANK members from the MP-AID. This action is appropriate if the existing RIM has been replaced by another RIM for which a new panel interface has been activated. To save space on the MP-AID, ASG recommends deleting the EXECUTIVE and INFOBANK members of an existing RIM using panel TA75000.

You can remove UDS-TABLE members from the MP-AID using the MP-AID DELETE command.

For details of the MP-AID DELETE command, refer to *ASG-Manager Products Systems Administrator's Manual*.

The UX COMMAND

Use the UX command to enable ToolSet SERVICES.

UX Command Syntax

```
>>──UX name mp-name──┬─ALL──────┬──┬──;──┬──────────────────>
                     ├─RULE010──┤  └──.──┘
                     ├─RULE020──┤
                     ├─RULE030──┤
                     ├─RULE040──┤
                     ├─RULE050──┤
                     ├─RULE070──┤
                     └─RULE080──┘
```

where:

name is a HIERARCHY member name

mp-name is the name specified in the MPAID-NAME clause of the HIERARCHY member.

6 Customizing the Environment

This chapter includes these sections:

Global Variables Defined in ITEM Members 106

Global Variables Defined in SEXEC Members 107
Customizing Functional Areas Using Global Variables 107

Customizing the Assisted Update 109
Standard String Delimiter: MDG_STADEL 109
Secondary String Delimiter: MDG_SECDEL 109
Translation of Alphabetic Characters: MDG_UPDLOW 110
Translation of Internal Keywords: MDG_MIXED1, MDG_MIXED2 110
Clause Separator: MDG_ATTSEP 111
Line Erase Character(s): MDG_DELSTR 112
Blank String Character(s): MDG_BLASTR 112
Keyword Indicator: MDG_UPDHEAD 112
Offset for Member Type Alias: MDG_SYMOFF 113
Specifying Prompt Formats 113
Standard Prompt: MDG_SKSTR2 114
Time Prompt: MDG_SKSTR3 114
Date Prompt: MDG_SKSTR4 114
Alias Prompt: MDG_SKSTR5 115
Compulsory Input Prompt: MDG_SKSTR6 115
Selection Prompt: MDG_SKSTR7 115
Line Protection Character: MDG_LINE_PROTECTION_CHAR 116
Hexadecimal Code of Line Protection Character: MDG_LINE_PROTECTION_CODE 117
Formatting Process Indicator: MDG_AUPD_AMEND and Formatting Process Bypass Array: MDG_AUPD_AMEND_EXCLUDE (N) 117

Customizing the Command Interface 118
Autoskip Feature: MDG_MMR_SET_AUTOSKIP 118
Buffer Limit: MDG_MMR_SET_BUFFER_LIMIT 118
Retention of Lookaside Buffers: MDG_MMR_SET_LOOKASIDE_RETENTION 119
Retention of Line Commands: MDG_MMR_SET_LINEAR_RETENTION 119
Condition for Update Output: MDG_MMR_SET_UPDATE_OUTPUT 120
Position of Line Command Area: MDG_MMR_SET_LINE_COMMAND 120
Position of Command Area: MDG_MMR_SET_COMMAND_LINE 121
Output Line Limit: MDG_MMR_SET_OUTPUT_LINES 121
Panel Limits: MDG_MMR_SET_PANEL_LIMITS 122

Customizing the Panel Interface 123
Position of the Command Area for a Single Panel Type: MDG_MMR_CX_CMD_LINE(N) . 123

Panel Type for which a Command Area is to be Generated: MDG_MMR_CX_CMD_TYPE(N) . 124
Control Whether Panel Invokes the Panel Display Exit (EC0995): MDG_GEN_PANEL_EXIT . . 125
Character that Marks an Input Field on a Panel: MDG_TABLE_FIELD_CHAR 125
Character that Marks the Command Area on a Panel: MDG_COMMAND_LINE_CHAR. . . . 125
Character that Marks the Line Command Area on a Panel: MDG_LINE_COMMAND_CHAR . . 126
Enable/Disable Automatic Logoff from Manager Software Products: MDG_LOGOFF 126
Number of Columns a Member Name is to be Indented in a Relationship Display: MDG_STINC. 126
Maximum Depth for the USA and REFA Line Commands: MDG_STMAX 127
Separator Between Member Name and Level Number in a Relationship Display: MDG_STSEP . 127
Maximum Number of Columns of a Matrix Displayed Online: MDG_MATRIX_SIZE_ONLINE 127
Maximum Number of Columns of a Matrix Processed in Batch: MDG_MATRIX_SIZE_BATCH 128
User-Definable Areas on Panel: MDG_USER_AREA_1 and MDG_USER_AREA_2. 128

Customizing the Documentation Functions 129
Enable or Disable Copy Function of DCUPD Command: MDG_DOKINC 130
Clause Defining the Body of a Document: MPR_EA60_DBODY. 131
Clause Defining the Heading of a Document: MPR_EA60_HEADING 131
Enable or Disable Automatic Composition of Complex Documentation from Several Levels of Sub-documents via the ??INCLUDE Command: MPR_EA60_DECOMPOSE. 132
Enable or Disable Automatic Numbering of Headings: MPR_EA60_INDEX. 132

Customizing Naming Conventions of Members. 133
Wildcard for Maximum Length of a Member Name: MDG_NAMEOL 133
Wildcard for Exact Length of a Member Name: MDG_NAMEON 133
Wildcard for Minimum Length of a Member Name: MDG_NAMSOL. 134
Wildcard for a Mandatory Alphanumeric or Special Value: MDG_NAMJOK 134
Wildcard for a Numeric Value: MDG_NAMNUM 134
Wildcard for the Variable Part of a Member Name: MDG_NAMVAR 135
Wildcard for any Number of Optional Alpanumeric Values: MDG_NAMOPT 135
Enable/Disable Assisted Update for Existing Members with Invalid Naming Convention: MDG_NAM_OLD. 136
Specify Existing Member Type(s) with Invalid Naming Convention for which the Assisted Update is Enabled or Disabled: MDG_NAM_OLD_MEM(N). 136
Specify Standard Names and Abbreviations for Repository Members: MDG_NAM_STD_NAME(N) and MDG_NAM_STD_ABBREV(N) 137
Enforce Standard Member Names in Assisted Update: MDG_NAM_ENFORCE. 138
Enforce Naming Conventions for Dummy Members: MDG_NAM_NEW 139
Enable/Disable Naming Conventions throughout the Repository: MDG_NAMTST. 139

Customizing the Retain Options 140

Customizing the Workbench Design Area 141
Enable or Disable ITEM Member Check: MDG_WBDA_ITEM_CHECK 141
Enable or Disable Replacement of Substring in Naming Convention of ITEM Members: MDG_WBDA_ITEM_REPLACE 141

Indicator of Substring to be Replaced in Naming Convention of ITEM Members: MDG_WBDA_SWITCH_PRSU_IT 142
Existing Prefix of ITEM Member Name that is to be Replaced: MDG_WBDA_ITEM_PREF_OLD 142
New Prefix that Replaces Existing Prefix of ITEM Member Name: MDG_WBDA_ITEM_PREF_NEW 143
Existing Suffix of ITEM Member Name that is to be Replaced: MDG_WBDA_ITEM_SUFF_OLD 143
New Suffix that Replaces Existing Suffix of ITEM Member Name: MDG_WBDA_ITEM_SUFF_NEW 144
Character that Initiates the Generation of a Default Identifier Name for the Data Element of an Entity: MDG_WBDA_RHSPRE 144
Name of User-defined Member Type Defining an Object of a DB2 or SQL/DS Database System: MDG_WBDA_TABLE_TYPE(N) 145
Indicator of Naming Conventions for Members Generated from Objects of a DB2 or SQL/DS Database System in the WBDA: MDG_WBDA_TABLE_PRSU(N). 147
Name of User-defined Executive Routine: MDG_WBDA_NAMING_EXIT 148

Activating User Exits for Toolset Services 149

Customizing Return From Buffers 150

Customizing Life Cycle Services (LCS) 151
Customizing Member Types Relevant for Life Cycle Services 151
Customizing Clauses of Member Types Relevant for LifeCycle SERVICES 154
Customizing Relationships Between Member Types Relevant for LifeCycle Services 158
Customizing Panels Used Under Life Cycle Services. 161
Customizing Project Management Functions of Life Cycle Services 163
Customizing Clauses Defining the Duration of a Task or a Project 166
Activating User Exits for Life Cycle Services. 168

Customizing - Miscellaneous 169

Customizing means the adaptation of an ASG-supplied environment to your corporate requirements.

By customizing the environment you define:

- Access to the repository
- The appearance of the user interface
- The processing of information contained or to be stored in the repository

Access to the repository is controlled by Logon Profiles and Global Profiles. Use a LOGON-PROFILE member, for instance to define the logon-identification, the password and the authority for a user. Use a GLOBAL-PROFILE member, for instance to route a group of users to the correct repository in a multiple repository environment. For details of Logon Profiles and Global Profiles, refer to the *ASG-Manager Products Systems Administrator's Manual*.

The appearance of the user interface and the processing of information in the repository is controlled by global variables and user exits. For details of user exits, refer to Chapter 7, "User Exits," on page 171.

This chapter describes how to customize the environment by changing the setting of ASG-supplied global variables.

ASG-supplied global variables are defined in ITEM members or in reserved SEXEC members of the Administration Repository.

Global Variables Defined in ITEM Members

Global variables that can be changed interactively using the Change your MethodManager profile function of the panel interface are defined in ITEM members. These ITEM members are combined in groups that make the general profile. Each group is used to customize a certain functional area of the environment.

A restricted number of global variables can be changed by common users as well. These variables contain the CATALOG entry, `LEVEL=USER` in their ITEM member definition. Only those variables are displayed for common users who access the Change your MethodManager profile function. The full range of variables displays only for Systems Administrators and Controllers.

A changed setting becomes active once you press Enter.

Whenever the setting is changed for the first time a USER-MEMBER called MMRUSER will be added to the MP-AID. The USER-MEMBER contains the individual setting of the global variables for a certain user. The individual setting is saved across sessions and will be intact when a user logs on again.

To change the defaults of ASG-supplied global variables:

- Update the ITEM member that defines the global variable.
- Define the new default value in the CONTENTS IS clause.
- File the ITEM member.
- Enter `PX ALL;` in a batch job that activates the changed profile.

To insert user-defined global variables in the general profile use the ASG-supplied GROUP member GR-MMR-USER. To do so:

- Define your global variable in an ITEM member
- File the ITEM member
- Specify the repository name of the ITEM member in the CONTAINS clause of the GROUP member GR-MMR-USER
- Enter PX ALL; in a batch job that activates the changed profile

For details of the ITEM member definition, refer to Chapter 9, "Member Types," on page 215.

Note:

ASG-supplied global variables use the naming convention MDG_. Please use another naming convention, for instance UDV_ for user-defined global variables.

Global Variables Defined in SEXEC Members

Global variables that cannot be changed interactively via the panel interface for technical reasons are defined in the SEXEC members EC1060 (MP-AID name = EASY-USER) and EH8000.

These variables can only be changed by Systems Administrators in the relevant SEXEC member.

When changing the value of a global variable in an SEXEC member, make sure that the new value is enclosed in literals.

To activate a changed setting:

- File the SEXEC member that contains the changed variable(s).
- Construct the SEXEC member onto the MP-AID using the CX command.
- Exit MethodManager.
- Access MethodManager again using the LCS or TSS command.

Note:

Do not define user-defined global variables in SEXEC members, because global variables currently contained in SEXEC members will be integrated into the general profile with the next release of the software. Define your user-defined global variables in ITEM members and insert the ITEM member in the GROUP member GR-MMR-USER of the general profile.

Customizing Functional Areas Using Global Variables

Table 6.1 shows which functional areas can be customized using global variables and where in the Administration repository these variables are set.

6.1 Functional Areas that can be Customized Using Global Variables.

Global Variables	Set in General Profile via TSS Change your MMR profile function	Set in SEXEC EC1060	Set in SEXEC EH8000
For customizing the listed functional areas	• assisted update • command interface • LCS clauses • LCS duration • LCS member-types • LCS panels • LCS project management • LCS relationships • LCS user exits • naming conventions • panel interface • retain options • TSS user exits • WBDA • miscellaneous	• assisted update • documentation functions • naming conventions • panel interface • WPDA • buffer return • miscellaneous	• documentation functions

In the following sections the global variables for customizing a functional area are described in a table. The last line of the table (Location) informs you where a global variable is set in the repository. The term *general profile* refers to ToolSet SERVICES Change your MethodManager profile function. Otherwise the name of the SEXEC member containing the global variable is specified.

Customizing the Assisted Update

To customize different features of the assisted update use the global variables described in this section.

Standard String Delimiter: MDG_STADEL

Use MDG_STADEL to specify the standard string delimiter. When a member has been updated in the assisted update the entries of certain clauses (for instance of the CATALOG clause) have to be enclosed in standard string delimiters. This happens automatically when the member is filed in the repository. When it is called back into the assisted update the member is reformatted so that the standard string delimiters do not appear. See "Example 1" on page 109.

Variable Name:	MDG_STADEL
Variable Length:	1
Valid Values:	single quote (') or double quote (")
Default Value:	single quote (')
Location:	general profile

Secondary String Delimiter: MDG_SECDEL

Use MDG_SECDEL to specify a quote that can be used in a string entered in a clause of the assisted update. It enables the system to distinguish between quotes used internally for delimiting an entry and quotes which are an integral part of an entry. See "Example 1" on page 109.

Example 1

MDG_STADEL is set to single quotes (') and
MDG_SECDEL is set to double quotes (").

Suppose you are in the assisted update and enter the CATALOG classification FRED’S ITEM:

- It is filed as 'FRED"S ITEM' (repository format)
- It displays as FRED"S ITEM (assisted update format).

Now you change the setting of MDG_STADEL to double quotes (") and MDG_SECDEL to single quotes (').

Alternatively suppose you are in the assisted update, and enter the CATALOG classification HUGO'S ITEM:

- It is filed as `"HUGO'S ITEM"` (repository format)
- It displays as `HUGO'S ITEM` (assisted update format).

Variable Name:	MDG_SECDEL
Variable Length:	1
Valid Values:	single quote (') or double quote (")
Default Value:	double quote (")
Location:	general profile

Translation of Alphabetic Characters: MDG_UPDLOW

Use MDG_UPDLOW to specify whether lower case alphabetic characters entered in the assisted update are translated to upper case. This might be especially important for clauses which contain help or descriptive text. Valid values:

- Y: mixed case active, lower case characters are not translated to upper case when the member is filed
- Any other value or none: mixed case inactive, lower case characters are translated to upper case when the member is filed

Variable Name:	MDG_UPDLOW
Variable Length:	1
Valid Values:	Y, any other value or none
Default Value:	Y
Location:	general profile

Translation of Internal Keywords: MDG_MIXED1, MDG_MIXED2

Use MDG_MIXED1 and MDG_MIXED2 to specify whether lower case alphabetic characters entered in the assisted update are translated to upper case even if MDG_UPDLOW is set to Y. Valid values of MDG_MIXED1 and MDG_MIXED2:

- Y: special treatment for internal keywords active
- Any other value or none: special treatment for internal keywords inactive

Translation is important for certain entries which are used as internal keywords and which would otherwise not be recognized by the system. See "Example 2" on page 111.

Note:

The variable MDG_MIXED1 only impacts the process if MDG_UPDLOW is set to Y. Otherwise it is ignored regardless of its settings.

Example 2

A GROUP member contains these entries in its CONTAINS clause:

```
-------------------------------------------------------
*CONTAINS
IT-ACC-NO
else IT-INT-REF
-------------------------------------------------------
```

If MDG_UPDLOW is set to Y and MDG_MIXED1 is set to N the else is not translated to upper case characters, nor will the GROUP member encode successfully, because else IT-INT-REF is regarded as an invalid member name by the system.

The system will only be able to identify else as an internal keyword if MDG_MIXED1 is set to Y.

Variable Name:	MDG_MIXED1, MDG_MIXED2
Variable Length:	1
Valid Values:	Y, any other value or none
Default Value:	Y
Location:	general profile

Clause Separator: MDG_ATTSEP

Use MDG_ATTSEP to specify a string separating different clauses of a member definition in the assisted update.

Variable Name:	MDG_ATTSEP
Variable Length:	10
Valid Values:	any alphanumeric and special characters
Default Value:	broken line (----------)
Location:	EC1060

Line Erase Character(s): MDG_DELSTR

Use MDG_DELSTR to specify a string that causes the deletion of each line containing this string when the member is filed in the repository.

Note:

When defining the string of the line erase character in SEXEC EC1060, you must use concatenation symbols (hex code 4F). If you call the SEXEC member EC1060 into the assisted update, change the value of MDG_DELSTR without using concatenation symbols and file the SEXEC member in the repository, the defined line erase character will delete itself and cause unpredictable results.

Variable Name:	MDG_DELSTR
Variable Length:	3
Valid Values:	any alphanumeric and special characters
Default Value:	'??.'
Location:	EC1060

Blank String Character(s): MDG_BLASTR

Use MDG_BLASTR to specify a string that is set to blanks when the member is filed in the repository.

Variable Name:	MDG_BLASTR
Variable Length:	4
Valid Values:	any alphanumeric and special characters
Default Value:	' ?.'
Location:	EC1060

Keyword Indicator: MDG_UPDHEAD

Use MDG_UPDHEAD to specify the character that is displayed immediately before the keyword of a clause in the assisted update.

Variable Name:	MDG_UPDHEAD
Variable Length:	1
Valid Values:	any special character
Default Value:	'*'
Location:	EC1060

Offset for Member Type Alias: MDG_SYMOFF

Use MDG_SYMOFF to specify a column in the first line of the assisted update where the member type alias displays.

Variable Name:	MDG_SYMOFF
Variable Length:	2
Valid Values:	any one or two-digit integer (maximum = 70)
Default Value:	69
Location:	EC1060

Specifying Prompt Formats

Use the following variables MDG_SKSTR2 to MDG_SKSTR7 (10 - 15) to specify the format of prompts which are displayed below the keywords of the clauses in the assisted update.

The prompt indicates to the user what sort of entry is required for a certain clause in the assisted update.

Which prompt is displayed for a clause in the assisted update is defined by the SKELETON-CODE or SKELETON-TEXT of the ATTRIBUTE-TYPE member defining the clause.

When specifying the prompt formats make sure that prompts, which are used for clauses with optional entries, contain the line erase character (set in MDG_DELSTR) as a substring. If the prompts are not overwritten by entries, the line erase character ensures that clauses without entries are deleted when the member is filed in the repository.

Note:

Use concatenation symbols when defining prompts which contain the line erase character as a substring. Otherwise the line erase character will delete the prompts when the SEXEC member EC1060 is filed.

Standard Prompt: MDG_SKSTR2

Use MDG_SKSTR2 to specify a prompt that can be used for any clause.

Variable Name:	MDG_SKSTR2
Variable Length:	3
Valid Values:	any alphanumeric and special character
Default Value:	'??.'
Location:	EC1060

Time Prompt: MDG_SKSTR3

Use MDG_SKSTR3 to specify a prompt for clauses defining the time.

Variable Name:	MDG_SKSTR3
Variable Length:	5
Valid Values:	any alphanumeric and special character
Default Value:	'??.??'
Location:	EC1060

Date Prompt: MDG_SKSTR4

Use MDG_SKSTR4 to specify a prompt for clauses defining a date.

Variable Name:	MDG_SKSTR4
Variable Length:	12
Valid Values:	any alphanumeric and special character
Default Value:	'??.??.??'
Location:	EC1060

Alias Prompt: MDG_SKSTR5

Use MDG_SKSTR5 to specify a prompt for the ALIAS clause.

Variable Name:	MDG_SKSTR5
Variable Length:	10
Valid Values:	any alphanumeric and special character
Default Value:	' ?. ??.'
Location:	EC1060

Compulsory Input Prompt: MDG_SKSTR6

Use MDG_SKSTR6 to specify a prompt for clauses with compulsory input.

Note:

The default value does not contain the line erase character as a substring. Therefore the user has to overwrite the prompt in the assisted update to file the member.

Variable Name:	MDG_SKSTR6
Variable Length:	5
Valid Values:	any alphanumeric and special character
Default Value:	'?XXXX'
Location:	EC1060

Selection Prompt: MDG_SKSTR7

Use MDG_SKSTR7 to specify a prompt that is to be deleted to select a predefined keyword in a clause.

The prompts and the keywords have to be defined in the SKELETON-TEXT clause of the ATTRIBUTE-TYPE member defining the clause. For details of the ATTRIBUTE-TYPE member definition, refer to "ATTRIBUTE-TYPE" on page 220.

When in the assisted update the user just needs to delete the prompt to select the relevant keyword. See "Example 3" on page 116.

Example 3

```
-------------------------------------------------------
*STANDING            Quality or status of the definition
L??.  D
P
L??.  A
L??.  S

-------------------------------------------------------
```

Variable Name:	MDG_SKSTR7
Variable Length:	5
Valid Values:	any alphanumeric and special character
Default Value:	'L??. '
Location:	EC1060

Line Protection Character: MDG_LINE_PROTECTION_CHAR

Use MDG_LINE_PROTECTION_CHAR to specify a character that causes the protection of a line in the assisted update. MDG_LINE_PROTECTION_CHAR ensures that the predefined update skeleton cannot be overwritten by the user.

Variable Name:	MDG_LINE_PROTECTION_CHAR
Variable Length:	1
Valid Values:	any alphanumeric and special character
Default Value:	a non-printable character
Location:	EC1060

Hexadecimal Code of Line Protection Character: MDG_LINE_PROTECTION_CODE

Use MDG_LINE_PROTECTION_CODE to specify the hexadecimal code of the line protection character specified in MDG_LINE_PROTECTION_CHAR.

Note:

MDG_LINE_PROTECTION_CHAR and MDG_LINE_PROTECTION_CODE must be set in conjunction.

Variable Name:	MDG_LINE_PROTECTION_CODE
Variable Length:	2
Valid Values:	hexadecimal code of specified line protection character
Default Value:	51
Location:	EC1060

Formatting Process Indicator: MDG_AUPD_AMEND and Formatting Process Bypass Array: MDG_AUPD_AMEND_EXCLUDE (N)

The assisted update normally uses the AMEND command internally to format the member's source before it is displayed in the assisted update buffer. This ensures that the source is in a standardized format, facilitating the generation of a correct assisted update buffer. Under certain circumstances, however, this may not be desirable. These variables allow the bypassing of AMEND processing.

Use MDG_AUPD_AMEND to specify whether or not the member source is to undergo AMEND processing. The valid values are as follows:

Y =	AMEND is used as normal
N =	AMEND is not used
M =	The array MDG_AUPD_AMEND_EXCLUDE is searched for the member-type and, if found, AMEND is not used on members of this type.

Variable Name:	MDG_AUPD_AMEND_	MDG_AUPD_AMEND_EXCLUDE (N)
Variable Length:	1	255
Valid Values:	Y, N or M	any valid member type
Default Value:	Y	null
Location:	EC1060	EC1060

Customizing the Command Interface

Use the global variables described in this section to customize different features of the command interface.

Autoskip Feature: MDG_MMR_SET_AUTOSKIP

Use MDG_MMR_SET_AUTOSKIP to specify whether a line containing a line command becomes the current line when the line command has been executed. Valid values:

- ON: the line in which the command was entered becomes the current line
- OFF: the buffer is redisplayed exactly as it was when the command was given

Variable Name:	MDG_MMR_SET_AUTOSKIP
Variable Length:	3
Valid Values:	ON or OFF
Default Value:	OFF
Location:	general profile

Buffer Limit: MDG_MMR_SET_BUFFER_LIMIT

Use MDG_MMR_SET_BUFFER_LIMIT to specify the maximum number of buffers available to each user.

Variable Name:	MDG_MMR_SET_BUFFER_LIMIT
Variable Length:	3
Valid Values:	any one or two-digit integer (maximum = 999)
Default Value:	99
Location:	general profile

Retention of Lookaside Buffers: MDG_MMR_SET_LOOKASIDE_RETENTION

Use MDG_MMR_SET_LOOKASIDE_RETENTION to specify whether Lookaside Buffers are to be retained up to the maximum buffer count (set in MDG_MMR_SET_BUFFER_LIMIT) despite subsequent EDIT or UPDATE commands. Valid values:

- ON: any current lookaside buffers are retained when you enter an EDIT or UPDATE buffer
- OFF: any current lookaside buffers are automatically deleted when you enter an EDIT or UPDATE command

Variable Name: MDG_MMR_SET_LOOKASIDE_RETENTION

Variable Length: 3

Valid Values: ON or OFF

Default Value: ON

Location: general profile

Retention of Line Commands: MDG_MMR_SET_LINEAR_RETENTION

Use MDG_MMR_SET_LINEAR_RETENTION to specify whether a line command is retained in the Line Command Area after the command has been executed. Valid values:

- ON: subsequent line command is retained and a terminator inserted to show that the command has been executed
- OFF: subsequent line command is automatically deleted from the Line Command Area after execution

Variable Name: MDG_MMR_SET_LINEAR_RETENTION

Variable Length: 3

Valid Values: ON or OFF

Default Value: ON

Location: general profile

Condition for Update Output: MDG_MMR_SET_UPDATE_OUTPUT

Use MDG_MMR_SET_UPDATE_OUTPUT to specify the condition in which the updated source record of a member and any messages relating to it may be displayed in a Lookaside Buffer when the member is filed. Valid conditions:

- LONG: output whenever a member is filed
- WARN: output only when execution generates warnings or errors
- ERROR: output only when execution generates errors

Variable Name:	MDG_MMR_SET_UPDATE_OUTPUT
Variable Length:	5
Valid Values:	LONG, WARN or ERROR
Default Value:	ERROR
Location:	general profile

Position of Line Command Area: MDG_MMR_SET_LINE_COMMAND

Use MDG_MMR_SET_LINE_COMMAND to specify the position of the Line Command Area on the screen. Valid values:

- LEFT: positions the Line Command Area on the left of the screen
- RIGHT: positions the Line Command Area on the right of the screen
- OFF: removes the Line Command Area from the screen

Note:

The setting of MDG_MMR_SET_LINE_COMMAND determines the position of the Line Command Area in the command interface. The setting of this variable has no impact on the position of the Line Command Area in the panel interface.

Variable Name:	MDG_MMR_SET_LINE_COMMAND
Variable Length:	5
Valid Values:	LEFT, RIGHT or OFF
Default Value:	LEFT
Location:	general profile

Position of Command Area: MDG_MMR_SET_COMMAND_LINE

Use MDG_MMR_SET_COMMAND_LINE to specify the position of the Command Area on the screen. Valid values:

- TOP: positions the Command Area to the top of the screen
- BOTTOM: positions the Command Area to the bottom of the screen

Note:

The setting of MDG_MMR_SET_COMMAND_LINE determines the position of the command area in the command interface. The setting of this variable has no impact on the position of the command area in the panel interface.

Variable Name:	MDG_MMR_SET_COMMAND_LINE
Variable Length:	6
Valid Values:	TOP or BOTTOM
Default Value:	TOP
Location:	general profile

Output Line Limit: MDG_MMR_SET_OUTPUT_LINES

Use MDG_MMR_SET_OUTPUT_LINES to specify the maximum number of lines of output that can be generated in any output buffer.

Variable Name:	MDG_MMR_SET_OUTPUT_LINES
Variable Length:	5
Valid Values:	any one to five-digit integer (maximum = 99999)
Default Value:	1000
Location:	general profile

Panel Limits: MDG_MMR_SET_PANEL_LIMITS

Use MDG_MMR_SET_PANEL_LIMITS to enforce output line and EXCP limits in MethodManager panels. Valid values:

- ON which enforces output line and EXCP limits
- OFF which ignores output line and EXCP limits

Note:

The restriction applies to commands issued via ARRAYGEN and EXTRACT. Both of these commands are used when panels are processed internally.

Variable Name:	MDG_MMR_SET_PANEL_LIMITS
Variable Length:	3
Valid Values:	ON or OFF
Default Value:	OFF
Location:	general profile

Customizing the Panel Interface

To customize different features of the panel interface use the following global variables.

Position of the Command Area for a Single Panel Type: MDG_MMR_CX_CMD_LINE(N)

Use MDG_MMR_CX_CMD_LINE(N) to specify the position of the Command Area for a panel type specified in MDG_MMR_CX_CMD_TYPE(N). Valid positions:

- BOTTOM: at the bottom of the panel
- TOP: at the top of the panel
- LINE nn: on a specified line
- NO: suppresses the generation of a command area

Variable Name:	MDG_MMR_CX_CMD_LINE(N)
Variable Length:	7
Valid Values:	BOTTOM, TOP, LINE nn, NO
Default Values:	MDG_MMR_CX_CMD_LINE(1) = TOP
	MDG_MMR_CX_CMD_LINE(2) = TOP
	MDG_MMR_CX_CMD_LINE(3) = TOP
	MDG_MMR_CX_CMD_LINE(4) = TOP
Location:	EC1060

Note:

The global variables MDG_MMR_CX_CMD_LINE(N) and MDG_MMR_CX_CMD_TYPE(N) must be set in conjunction.

Panel Type for which a Command Area is to be Generated: MDG_MMR_CX_CMD_TYPE(N)

Use MDG_MMR_CX_CMD_TYPE(N) to specify the panel type for which a Command Area is to be generated. The position of the Command Area for the specified panel type is set in the global variable MDG_MMR_CX_CMD_LINE(N). Valid values:

- MENU: specifies a Command Area for menu panels
- INPUT: specifies a Command Area for input panels
- LIST: specifies a Command Area for list panels
- OUTPUT: specifies a Command Area for output panels

Variable Name:	MDG_MMR_CX_CMD_TYPE(N)
Variable Length:	6
Valid Values:	MENU, INPUT, LIST, OUTPUT
Default Values:	MDG_MMR_CX_CMD_TYPE(1) = MENU
	MDG_MMR_CX_CMD_TYPE(2) = INPUT
	MDG_MMR_CX_CMD_TYPE(3) = LIST
	MDG_MMR_CX_CMD_TYPE(4) = OUTPUT
Location:	EC1060

To change the setting of the Command Area for one panel, use the COMMAND-LINE clause of the FMT-SCREEN member defining the panel: the settings of the global variables MDG_MMR_CX_CMD_LINE and MDG_MMR_CX_CMD_TYPE will be ignored for the current FMT-SCREEN member. For details of the FMT-SCREEN member, refer to Chapter 9, "Member Types," on page 215.

Control Whether Panel Invokes the Panel Display Exit (EC0995): MDG_GEN_PANEL_EXIT

Use MDG_GEN_PANEL_EXIT to control whether or not the CX command enables a panel to invoke the Panel Display exit (EC0995) when the panel displays.

(See "Tailoring the Panel Display" on page 184 for a description of the Panel Display exit.)

Variable Name:	MDG_GEN_PANEL_EXIT
Variable Length:	1
Valid Values:	Y or N
Default Value:	N
Location:	EC1060

Character that Marks an Input Field on a Panel: MDG_TABLE_FIELD_CHAR

Use MDG_TABLE_FIELD_CHAR to specify a character that marks an input field on a panel.

Variable Name:	MDG_TABLE_FIELD_CHAR
Variable Length:	1
Valid Values:	any special character that is not used as leading/trailing character in any data to be entered
Default Value:	underscore (_)
Location:	general profile

Character that Marks the Command Area on a Panel: MDG_COMMAND_LINE_CHAR

Use MDG_COMMAND_LINE_CHAR to specify a character that marks the Command Area on a panel.

Variable Name:	MDG_COMMAND_LINE_CHAR
Variable Length:	1
Valid Values:	any special character that is not used as leading/trailing character in any data to be entered
Default Value:	blank ( )
Location:	general profile

Character that Marks the Line Command Area on a Panel: MDG_LINE_COMMAND_CHAR

Use MDG_LINE_COMMAND_CHAR to specify a character that marks the line command area on a panel.

Variable Name:	MDG_LINE_COMMAND_CHAR
Variable Length:	1
Valid Values:	any special character that is not used as leading or trailing character in any data to be entered
Default Value:	equals (=)
Location:	general profile

Enable/Disable Automatic Logoff from Manager Software Products: MDG_LOGOFF

Use MDG_LOGOFF to specify whether an automatic logoff from Manager Products software occurs after leaving MethodManager. Valid values:

- Y: automatic logoff enabled
- Any other value or none: automatic logoff disabled

Variable Name:	MDG_LOGOFF
Variable Length:	1
Valid Values:	Y, any other value or none
Default Value:	none
Location:	general profile

Number of Columns a Member Name is to be Indented in a Relationship Display: MDG_STINC

Use MDG_STINC to specify the number of columns by which a member name is to be indented in a display showing the relationships between members.

Variable Name:	MDG_STINC
Variable Length:	1
Valid Values:	0 - 9
Default Value:	3
Location:	general profile

Maximum Depth for the USA and REFA Line Commands: MDG_STMAX

Use MDG_STMAX to specify the maximum depth for the USA and REFA line commands.

Variable Name:	MDG_STMAX
Variable Length:	2
Valid Values:	01 - 99
Default Value:	05
Location:	general profile

Separator Between Member Name and Level Number in a Relationship Display: MDG_STSEP

Use MDG_STSEP to specify the separator between the member name and level number in a display showing the relationships between members.

Variable Name:	MDG_STSEP
Variable Length:	1
Valid Values:	any special character
Default Value:	hyphen (-)
Location:	general profile

Use the following variables to tailor a matrix which displays the relationships between members of two KEPT-DATA lists.

Tailoring changes the display of panel TD45000 of ToolSet SERVICES (TSS) and of the MATRIX command of LifeCycle SERVICES (LCS).

Maximum Number of Columns of a Matrix Displayed Online: MDG_MATRIX_SIZE_ONLINE

Use MDG_MATRIX_SIZE_ONLINE to specify the maximum number of columns of a matrix display online.

Variable Name:	MDG_MATRIX_SIZE_ONLINE
Variable Length:	3
Valid Values:	any integer
Default Value:	254
Location:	general profile

Maximum Number of Columns of a Matrix Processed in Batch: MDG_MATRIX_SIZE_BATCH

Use MDG_MATRIX_SIZE_BATCH to specify the maximum number of columns of a matrix to be processed in batch for output to a printer.

Variable Name:	MDG_MATRIX_SIZE_BATCH
Variable Length:	3
Valid Values:	any integer
Default Value:	121
Location:	general profile

Note:

When changing the default, make sure that the specified integer does not exceed the maximum number of columns to which your printer is adjusted.

User-Definable Areas on Panel: MDG_USER_AREA_1 and MDG_USER_AREA_2

Use MDG_USER_AREA_1 and MDG_USER_AREA_2 to redefine the areas immediately above and below the MethodManager command line when it is positioned at the top of the panel. Only the first 78 characters should be used, since this is the panel width.

MDG_USER_AREA_1 defines the area immediately above the command line which typically shows the current repository and status details and, when displaying a LIST or OUTPUT panel, also shows line number information on the right. To ensure upward compatibility with previous releases, this variable is reset to null in SEXEC EC0950, which blanks this screen area. You should therefore use EC0950 to set the contents of MDG_USER_AREA_1.

MDG_USER_AREA_2 is set to null by default, but you may assign it a value using SEXEC EC1060 (MP-AID name EASY-USER) if the value is static (such as a name) or EC0950 if the value is dynamic (such as &TIME).

Variable Name:	MDG_USER_AREA_1 and MGD_USER_AREA_2
Variable Length:	78
Valid Values:	any
Default Value:	null
Location:	EC0950 or EC1060

Note:

Use the CX command to reconstruct any panels that contain user tailoring to the panel interface.

Customizing the Documentation Functions

To customize different features of the documentation functions use the global variables described in the following sections.

Note:

The global variables 2 through 5 use an old naming convention. Their naming convention will be changed to MDG_ with the next release.

Enable or Disable Copy Function of DCUPD Command: MDG_DOKINC

Use MDG_DOKINC to enable or disable the copy function of the DCUPD command. If active, the DCUPD command enables you to create a new DOCUMENT member and copy its CONTENTS clause information from an existing member. For details of the DCUPD command, refer to "DCUPD" on page 359. Valid values:

- Y: copy function of DCUPD command enabled
- Any other value or none: copy function of DCUPD command disabled. This means that you can still use the DCUPD command to create a new DOCUMENT member but you cannot copy information from an existing member into its CONTENTS clause even if the name of the existing member is specified in the DCUPD command

Note:

If you change the setting of MDG_DOKINC to disable the copy function of the DCUPD command this will have an impact on the :DCSTANDARD macro which internally uses DCUPD.

The :DCSTANDARD macro can be specified in a LIFE-CYCLE-OBJECT-TYPE member to list DOCUMENT members of the current project. For details of the :DCSTANDARD macro, refer to ":DCSTANDARD" on page 349. If :DCSTANDARD is specified in a LIFE-CYCLE-OBJECT-TYPE producing a deliverable, the line command S can be used to create a new DOCUMENT member. A standard form will be automatically inserted in the CONTENTS clause of the new DOCUMENT member, if MDG_DOKINC is set to Y and if the standard form is defined in a separate DOCUMENT member.

If you disable the copy function of the DCUPD command the :DCSTANDARD macro will still create a new DOCUMENT member but the standard form will not be inserted in the CONTENTS clause of the new member, even if defined in a separate DOCUMENT member.

Variable Name:	MDG_DOKINC
Variable Length:	1
Valid Values:	Y, any other value or none
Default Value:	Y
Location:	EC1060

Clause Defining the Body of a Document: MPR_EA60_DBODY

Use MPR_EA60_DBODY to specify the name of a clause that defines the body of a document. The body consists of the actual document text and of document commands and Manager Products commands.

The name of the clause has to be specified as defined in the IDENTIFIED-BY clause of its ATTRIBUTE-TYPE member. The attribute type must be TEXT or FREE-FORM-TEXT.

Variable Name:	MPR_EA60_DBODY
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	CONTENTS
Location:	EH8000

Clause Defining the Heading of a Document: MPR_EA60_HEADING

Use MPR_EA60_HEADING to specify the name of a clause that defines the heading of a document. The name of the clause has to be specified as defined in the IDENTIFIED-BY clause of its ATTRIBUTE-TYPE member. The attribute type can be TEXT, FREE-FORM-TEXT, or CHARACTER-STRING. If the attribute type is TEXT or FREE-FORM-TEXT the heading is generated from the first line of entries.

Variable Name:	MPR_EA60_HEADING
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	TITLE
Location:	EH8000

Enable or Disable Automatic Composition of Complex Documentation from Several Levels of Sub-documents via the ??INCLUDE Command: MPR_EA60_DECOMPOSE

Use MPR_EA60_DECOMPOSE to specify if a subordinate document is to be inserted in a higher document from where it is referenced via the ??INCLUDE command. Valid values:

- YES: enable automatic composition
- NO: disable automatic composition

Note:

If MPR_EA60_DECOMPOSE is set to NO, only the first level of subordinate documents will be inserted in the higher document from where they are referenced. If the subordinate documents themselves use further ??INCLUDE commands, these commands will be ignored.

Variable Name:	MPR_EA60_DECOMPOSE
Variable Length:	3
Valid Values:	YES, NO
Default Value:	YES
Location:	EH8000

Enable or Disable Automatic Numbering of Headings: MPR_EA60_INDEX

Use MPR_EA60_INDEX to specify if headings are to be numbered automatically in the document text and in the table of contents. Valid values are YES to enable automatic numbering or NO to disable automatic numbering.

Variable Name:	MPR_EA60_INDEX
Variable Length:	3
Valid Values:	YES, NO
Default Value:	YES
Location:	EH8000

Customizing Naming Conventions of Members

Use the following global variables to customize the naming conventions of repository members. Some global variables specify wildcards that indicate the length and the validity of a member name. They are used to define the naming convention of members in the NAMING clauses of MEMBER-TYPE and RELATIONSHIP-TYPE members. The naming conventions—usually a prefix followed by the specified wildcards—are displayed in list panels and indicate the structure of a valid member name.

Note:

When changing the defaults of the following global variables, make sure that the NAMING clauses of the MEMBER-TYPE and RELATIONSHIP-TYPE members are changed correspondingly.

Wildcard for Maximum Length of a Member Name: MDG_NAMEOL

Use MDG_NAMEOL to specify a character that indicates the maximum length of a member name. If a user enters a member name in a panel the name must not overwrite the specified character.

Variable Name:	MDG_NAMEOL
Variable Length:	1
Valid Values:	any special character
Default Value:	less than (<)
Location:	general profile

Wildcard for Exact Length of a Member Name: MDG_NAMEON

Use MDG_NAMEON to specify a character that indicates the exact length of a member name. If a user enters a member name in a panel the name must stop immediately before the specified character with no blank in between.

Variable Name:	MDG_NAMEON
Variable Length:	1
Valid Values:	any special character
Default Value:	equals (=)
Location:	general profile

Wildcard for Minimum Length of a Member Name: MDG_NAMSOL

Use MDG_NAMSOL to specify a character that indicates the minimum length of a member name. If a user enters a member name in a panel the name must overwrite the specified character.

Variable Name:	MDG_NAMSOL
Variable Length:	1
Valid Values:	any special character
Default Value:	greater than (>)
Location:	general profile

Wildcard for a Mandatory Alphanumeric or Special Value: MDG_NAMJOK

Use MDG_NAMJOK to specify a character that indicates that an alphanumeric or a special character must be entered.

Variable Name:	MDG_NAMJOK
Variable Length:	1
Valid Values:	any special character
Default Value:	underscore (_)
Location:	general profile

Wildcard for a Numeric Value: MDG_NAMNUM

Use MDG_NAMNUM to specify a character that indicates that only a numeric value can be entered.

Variable Name:	MDG_NAMNUM
Variable Length:	1
Valid Values:	any special charactER
Default Value:	hash (#)
Location:	general profile

Wildcard for the Variable Part of a Member Name: MDG_NAMVAR

Use MDG_NAMVAR to specify a character that indicates the variable part of a member name. The character is used to enclose a variable in the naming convention of a member.

Variable Name:	MDG_NAMVAR
Variable Length:	1
Valid Values:	any special character
Default Value:	colon (:)
Location:	general profile

Wildcard for any Number of Optional Alpanumeric Values: MDG_NAMOPT

Use MDG_NAMOPT to specify a character that indicates that:

- No value needs to be entered
- One alpanumeric value can be entered
- Several (maximum = 32 including the member type prefix) alpanumeric values can be entered

Note:

If only a prefix and this wildcard defines the naming convention of a member, the number of wildcards displayed on a panel does not represent the actual number of entries for the member name.

Variable Name:	MDG_NAMOPT
Variable Length:	1
Valid Values:	any special character
Default Value:	asterisk (*)
Location:	general profile

Use the following global variables to enable or disable the assisted update facility for all or a specified set of member types that use no valid naming conventions.

Enable/Disable Assisted Update for Existing Members with Invalid Naming Convention: MDG_NAM_OLD

Use MDG_NAM_OLD to enable or disable the assisted update for existing members with invalid naming convention. Valid values:

- ON: existing members with invalid naming convention can be updated using the assisted update
- OFF: existing members with invalid naming convention cannot be updated using the assisted update
- YES: only existing member types with invalid naming convention that are specified in MDG_NAM_OLD_MEM can be updated using the assisted update
- NO: only existing member types with invalid naming convention that are specified in MDG_NAM_OLD_MEM cannot be updated using the assisted update

Note:

If MDG_NAM_OLD is set to YES or NO, MDG_NAM_OLD_MEM must also be specified.

Variable Name:	MDG_NAM_OLD
Variable Length:	3
Valid Values:	ON, OFF, YES, NO
Default Value:	OFF
Location:	EC1060

Specify Existing Member Type(s) with Invalid Naming Convention for which the Assisted Update is Enabled or Disabled: MDG_NAM_OLD_MEM(N)

Use MDG_NAM_OLD_MEM(N) to specify the first ALIAS entry of an existing MEMBER-TYPE or RELATIONSHIP-TYPE member with an invalid naming convention for which the assisted update is to be enabled or disabled.

If several member types are to be specified, enter:

```
MDG_NAM_OLD_MEM(1) = A1
MDG_NAM_OLD_MEM(2) = A2
MDG_NAM_OLD_MEM(3) = B4
................    ..
MDG_NAM_OLD_MEM(N) = XY
```

Note:

The figure enclosed in brackets (N) indicates the element number of the array. The two-digit string enclosed in literals XY specifies the first ALIAS entry of the relevant MEMBER-TYPE or RELATIONSHIP-TYPE member.

Variable Name:	MDG_NAM_OLD_MEM
Variable Length:	2
Valid Values:	first ALIAS entry of a MEMBER-TYPE member
Default Value:	none
Location:	EC1060

Specify Standard Names and Abbreviations for Repository Members: MDG_NAM_STD_NAME(N) and MDG_NAM_STD_ABBREV(N)

Use MDG_NAM_STD_NAME(N) and MDG_NAM_STD_ABBREV(N) to specify standard name barrels and their abbreviations.

MDG_NAM_STD_NAME(N) holds the unabbreviated form of a name barrel if it has one or more abbreviations which are preferred.

MDG_NAM_STD_ABBREV(N) holds all acceptable abbreviations, separated by spaces, for the corresponding entry in MDG_NAM_STD_NAME(N).

For example:

```
MDG_NAM_STD_NAME(1)      = 'DEPARTMENT'
MDG_NAM_STD_NAME(2)      = 'NUMBER'
MDG_NAM_STD_NAME(3)      = 'ACCOUNT'
MDG_NAM_STD_ABBREV(1)    = 'DEPT'
MDG_NAM_STD_ABBREV(2)    = 'NUM NO'
MDG_NAM_STD_ABBEV(3)     = 'ACC'
```

These two variables comprise the standard abbreviation table. Separate tables for the data repository and administration repository can be defined in the executive-routines EL6007 and EL6008, which currently provide example standard abbreviation tables for DU016 and DU777 respectively.

Variable Name:	MDG_NAM_STD_NAME	MDG_NAM_STD_ABBREV
Variable Length:	up to 255	up to 255
Valid Values:	any barrel name	any barrel abbreviation
Default Value:	none	none
Location:	any executive routine (mpaid name EA13uds-name)	

Enforce Standard Member Names in Assisted Update: MDG_NAM_ENFORCE

Use MDG_NAM_ENFORCE to enforce standard names for new members created in assisted update. If MDG_NAM_ENFORCE = Y, any new member whose name does not conform to the standards held in the abbreviation table can be rejected by assisted update. This is achieved by special processing in Update Exit 2 (EC9992) which contains a coding example.

Variable Name:	MDG_NAM_ENFORCE
Variable Length:	1
Valid Values:	Y or N
Default Value:	Y
Location:	EC1060

Note:

See "Activating User Exits for Toolset Services" on page 149 for how to activate Update Exit 2.

Enforce Naming Conventions for Dummy Members: MDG_NAM_NEW

Use MDG_NAM_NEW to enforce naming conventions for dummy members. If MDG_NAM_NEW = Y, the naming convention is enforced. Consequently, if a dummy member created as a result of an assisted update does not have a valid name, the update is halted and an appropriate error message is issued. If MDG_NAM_NEW = N, the naming convention is ignored.

Variable Name:	MDG_NAM_NEW
Variable Length:	1
Valid Values:	Y or N
Default Value:	N
Location:	EC1060

Enable/Disable Naming Conventions throughout the Repository: MDG_NAMTST

Use MDG_NAMTST to specify whether naming conventions are to be enforced or ignored throughout the repository. Valid values are:

- Y: naming conventions are enforced throughout the repository
- N: naming conventions are ignored throughout the repository

If MDG_NAMTST is set to Y, naming conventions are enforced during assisted update for both new and existing members. However, existing members with invalid names can be updated using assisted update if the settings of the global variables MDG_NAM_OLD and MDG_NAM_OLD_MEM are suitable.

Variable Name:	MDG_NAMTST
Variable Length:	1
Valid Values:	Y or N
Default Value:	Y
Location:	EC1060

Customizing the Retain Options

To enable or disable the different :RETAIN macros use these global variables.

- MDG_RETAIN1 : enable/disable :RETAIN CLASS=1
- MDG_RETAIN2 : enable/disable :RETAIN CLASS=2
- MDG_RETAIN3 : enable/disable :RETAIN CLASS=3
- MDG_RETAIN4 : enable/disable :RETAIN CLASS=4
- MDG_RETAIN5 : enable/disable :RETAIN CLASS=5
- MDG_RETAIN6 : enable/disable :RETAIN CLASS=6
- MDG_RETAIN7 : enable/disable :RETAIN CLASS=7
- MDG_RETAIN8 : enable/disable :RETAIN CLASS=8
- MDG_RETAIN9 : enable/disable :RETAIN CLASS=9
- MDG_RETAIN_MFS : controls the retention of COMMAND members that display and manipulate panels
- MDG_RETAIN_MFR : controls COMMAND members that read and route panels
- MDG_RETAIN_EX : retains EXECUTIVE members constructed using the CX command
- MDG_RETAIN_ALL retains all the members retained by the above three macros

The variables described in this section have these common features:

Variable Length:	1
Valid Values:	Y or N
Default Value:	N
Location:	general profile (variables 1 through 9) EC1060 (variables 10 through 13)

The variables can be set to:

- Y: corresponding :RETAIN macro enabled
- N: corresponding :RETAIN macro disabled

Note:

The :RETAIN macro is used to specify the priority with which an Executive Routine is retained in virtual storage. For details of macros, refer to Chapter 8, "Macro Descriptions," on page 198.

Customizing the Workbench Design Area

To customize different features of the Workbench Design Area (WBDA) use the following global variables.

Enable or Disable ITEM Member Check: MDG_WBDA_ITEM_CHECK

Use MDG_WBDA_ITEM_CHECK to check if ITEM members which have been created in the WBDA already exist in the repository. The check is carried out during the preview stage. Valid values:

- 0: check disabled
- 1: check enabled

Variable Name:	MDG_WBDA_ITEM_CHECK
Variable Length:	1
Valid Values:	0 or 1
Default Value:	0
Location:	general profile

Enable or Disable Replacement of Substring in Naming Convention of ITEM Members: MDG_WBDA_ITEM_REPLACE

Use MDG_WBDA_ITEM_REPLACE to specify if a substring from the naming convention of an existing ITEM member is to be replaced by a new substring in the WBDA. Valid values:

- 0: replacement disabled
- 1: replacement enabled

Variable Name:	MDG_WBDA_ITEM_REPLACE
Variable Length:	1
Valid Values:	0 or 1
Default Value:	0
Location:	general profile

Indicator of Substring to be Replaced in Naming Convention of ITEM Members: MDG_WBDA_SWITCH_PRSU_IT

Use MDG_WBDA_SWITCH_PRSU_IT to specify a character that indicates if the prefix, the suffix or another string from the naming convention of an existing ITEM member is to be replaced. Valid values:

- P: replace the prefix
- S: replace the suffix
- U: replace another string

Note:

If U is specified, the global variable MDG_WBDA_NAMING_EXIT must be set to the name of a user-defined executive routine that carries out the replacement of the string.

Variable Name:	MDG_WBDA_SWITCH_PRSU_IT
Variable Length:	1
Valid Values:	P, S, U
Default Value:	P
Location:	EC1060

Existing Prefix of ITEM Member Name that is to be Replaced: MDG_WBDA_ITEM_PREF_OLD

Use MDG_WBDA_ITEM_PREF_OLD to specify the prefix from the naming convention of an existing ITEM member that is to be replaced by a new prefix.

Note:

MDG_WBDA_ITEM_REPLACE must be set to 1 and MDG_WBDA_SWITCH_PRSU_IT must be set to P if the prefix is to be replaced.

Variable Name:	MDG_WBDA_ITEM_PREF_OLD
Variable Length:	32
Valid Values:	any prefix as defined for ITEM members in the NAMING cause of their MEMBER-TYPE definition
Default Value:	IT-
Location:	general profile

New Prefix that Replaces Existing Prefix of ITEM Member Name: MDG_WBDA_ITEM_PREF_NEW

Use MDG_WBDA_ITEM_PREF_NEW to specify a new prefix that replaces the prefix from the naming convention of an existing ITEM member. The existing prefix that is to be replaced must be specified in the global variable MDG_WBDA_ITEM_PREF_OLD.

Note:

MDG_WBDA_ITEM_REPLACE must be set to 1 and MDG_WBDA_SWITCH_PRSU_IT must be set to P if the prefix is to be replaced.

Variable Name:	MDG_WBDA_ITEM_PREF_NEW
Variable Length:	32
Valid Values:	any prefix as defined for ITEM members in the NAMING clause of their MEMBER-TYPE definition
Default Value:	IT-

Existing Suffix of ITEM Member Name that is to be Replaced: MDG_WBDA_ITEM_SUFF_OLD

Use MDG_WBDA_ITEM_SUFF_OLD to specify the suffix from the naming convention of an existing ITEM member that is to be replaced by a new suffix.

Note:

MDG_WBDA_ITEM_REPLACE must be set to 1 and MDG_WBDA_SWITCH_PRSU_IT must be set to S if the suffix is to be replaced.

Variable Name:	MDG_WBDA_ITEM_SUFF_OLD
Variable Length:	32
Valid Values:	any suffix as defined for ITEM members in the NAMING clause of their MEMBER-TYPE definition
Default Value:	-IT
Location:	EC1060

New Suffix that Replaces Existing Suffix of ITEM Member Name: MDG_WBDA_ITEM_SUFF_NEW

Use MDG_WBDA_ITEM_SUFF_NEW to specify a new suffix that replaces the suffix from the naming convention of an existing ITEM member. The existing suffix that is to be replaced must be specified in the global variable MDG_WBDA_ITEM_SUFF_OLD.

Note:

MDG_WBDA_ITEM_REPLACE must be set to 1 and MDG_WBDA_SWITCH_PRSU_IT must be set to S if the suffix is to be replaced.

Variable Name:	MDG_WBDA_ITEM_SUFF_NEW
Variable Length:	32
Valid Values:	any suffix as defined for ITEM members in the NAMING clause of their MEMBER-TYPE definition
Default Value:	-IT
Location:	EC1060

Character that Initiates the Generation of a Default Identifier Name for the Data Element of an Entity: MDG_WBDA_RHSPRE

Use MDG_WBDA_RHSPRE to specify a character that initiates the generation of a default identifier name for the data element of an entity.

Note:

The value of MDG_WBDA_RHSPRE must be the same as the value of the LHSPRE keyword in the Data Modeling and Design installation macro LOPT1.

Variable Name:	MDG_WBDA_RHSPRE
Variable Length:	1
Valid Values:	any alphanumeric and special character
Default Value:	! (hexadecimal code 5A)
Location:	general profile

Change the setting of these global variables:

- If you use other than the ASG-supplied member types CONCEPTUAL-RECORD, CONCEPTUAL-RELATION, DB2-INDEX, DB2-TABLE, DB2-TBSPACE, DB2-VIEW, SQL-DBSPACE, SQL-TABLE, SQL-VIEW, and SQL-INDEX to define different objects of a DB2 or SQL/DS database system in the repository
- To specify the naming conventions of DB2 and SQL/DS members that have been generated in the Workbench Design Area (WBDA) and are to be stored in the repository.

Name of User-defined Member Type Defining an Object of a DB2 or SQL/DS Database System: MDG_WBDA_TABLE_TYPE(N)

Use MDG_WBDA_TABLE_TYPE(N) to specify the names of user-defined member types defining objects of a DB2 or SQL/DS database system in the repository. The names have to be specified as defined in the ENCODE-KEYWORDS clause of their MEMBER-TYPE member.

For each user-defined member type defining an object of a DB2 or SQL/DS database system a corresponding indicator must be specified in the global variable MDG_WBDA_TABLE_PRSU(N).

Note:

If you change the setting of MDG_WBDA_TABLE_TYPE(N) you must change some FORMAT members as well that define the contents and layout of reports from the Workbench Design Area (WBDA). For details of FORMAT members, refer to *ASG-DesignManager User Output*. If you specify a new member type name in MDG_WBDA_TABLE_TYPE(N) replace the existing name by the new member type name in the relevant FORMAT member. The update must be done in the corporate repository that uses the specified FORMAT members.

The following table shows which FORMAT member has to be tailored if the setting of MDG_WBDA_TABLE_TYPE(n) is changed:

Variable Name	Update FORMAT member
MDG_WBDA_TABLE_TYPE(1)	FMTW43030
MDG_WBDA_TABLE_TYPE(2)	FMTW43030
MDG_WBDA_TABLE_TYPE(3)	FMTW43030
MDG_WBDA_TABLE_TYPE(4)	FMTW43030
MDG_WBDA_TABLE_TYPE(5)	FMTW41000
MDG_WBDA_TABLE_TYPE(6)	FMTW42000
MDG_WBDA_TABLE_TYPE(7)	FMTW43050

Variable Name	Update FORMAT member
MDG_WBDA_TABLE_TYPE(8)	FMTW43050
MDG_WBDA_TABLE_TYPE(9)	FMTW43050
MDG_WBDA_TABLE_TYPE(10)	FMTW43050

Variable Name: MDG_WBDA_TABLE_TYPE(N)

Variable Length: 32

Valid Values: ENCODE-KEYWORDS entry of MEMBER-TYPE member

Default Values: MDG_WBDA_TABLE_TYPE(1) = DB2-TABLE
MDG_WBDA_TABLE_TYPE(2) = DB2-INDEX
MDG_WBDA_TABLE_TYPE(3) = DB2-VIEW
WBDA_TABLE_TYPE(4) = DB2-TBSPACE
MDG_WBDA_TABLE_TYPE(5) = CONCEPTUAL-RECORD
MDG_WBDA_TABLE_TYPE(6) = CONCEPTUAL-RELATION
MDG_WBDA_TABLE_TYPE(7) = SQL-TABLE
MDG_WBDA_TABLE_TYPE(8) = SQL-INDEX
MDG_WBDA_TABLE_TYPE(9) = SQL-VIEW
MDG_WBDA_TABLE_TYPE(10) = SQL-DBSPACE

Location EC1060

Note:

The global variables MDG_WBDA_TABLE_TYPE(N) and MDG_WBDA_TABLE_PRSU(N) must be set in conjunction.

Indicator of Naming Conventions for Members Generated from Objects of a DB2 or SQL/DS Database System in the WBDA: MDG_WBDA_TABLE_PRSU(N)

Use MDG_WBDA_TABLE_PRSU(N) to specify the naming conventions of members that have been generated from objects of a DB2 or SQL/DS database system in the Workbench Design Area (WBDA). The naming conventions of these members can be extended or modified before the members are filed in the repository.

The member types whose naming conventions are to be extended or modified are specified in the global variable MDG_WBDA_TABLE_TYPE(N).

Valid values of the global variable MDG_WBDA_TABLE_PRSU(N):

- N: no treatment of generated member name
- P: member type prefix added to generated member name
- S: member type suffix added to generated member name
- U: generated member name modified by user-defined executive routine

Note:

If U is specified, the global variable MDG_WBDA_NAMING_EXIT must be set to the name of a user-defined executive routine that carries out the modification of the member name before the member is filed in the repository.

If a prefix or suffix is to be added to the generated member name it will be derived from the NAMING clause of the MEMBER-TYPE member that defines the DB2 or SQL/DS object.

Variable Name:	MDG_WBDA_TABLE_PRSU(N)
Variable Length:	1
Valid Values:	N, P, S, U
Default Values:	MDG_WBDA_TABLE_PRSU(1) = P
	MDG_WBDA_TABLE_PRSU(2) = N
	MDG_WBDA_TABLE_PRSU(3) = N
	MDG_WBDA_TABLE_PRSU(4) = P
	MDG_WBDA_TABLE_PRSU(5) = P
	MDG_WBDA_TABLE_PRSU(6) = P
	MDG_WBDA_TABLE_PRSU(7) = P
	MDG_WBDA_TABLE_PRSU(8) = N

MDG_WBDA_TABLE_PRSU(9) = N

MDG_WBDA_TABLE_PRSU(10) = P

Location: EC1060

Name of User-defined Executive Routine: MDG_WBDA_NAMING_EXIT

Use MDG_WBDA_NAMING_EXIT to specify the MP-AID name of a user-defined executive routine that generates the required naming convention for a member.

The user-defined executive routine must be defined in an SEXEC or in an EXECUTIVE-ROUTINE member.

If the user-defined executive routine is called from the global variable MDG_WBDA_SWITCH_PRSU_IT to replace a substring in the naming convention of an ITEM member, the routine receives:

- The name of the ITEM member in P0
- The keyword ITEM in P1

If the user-defined executive routine is called from the global variable MDG_WBDA_TABLE_PRSU(N) to modify the naming conventions of members that have been generated from objects of a DB2 or SQL/DS database system in the WBDA, the routine receives:

- The generated member name in P0
- The generated member name in P1

In both cases the user-defined executive routine must return the complete modified member name in the global variable &G45.

Variable Name: MDG_WBDA_NAMING_EXIT

Variable Length: 10

Valid Values: MP-AID name of a user-defined executive routine

Default Value: none

Location: EC1060

Activating User Exits for Toolset Services

To activate user exits for ToolSet SERVICES (TSS) use the following global variables. For details of user exits, refer to Chapter 7, "User Exits," on page 171.

- MDG_NAMEXT1 : activate exit routine in SEXEC member EC9980
- MDG_NAMEXT2 : activate exit routine in SEXEC member EC9981
- MDG_UPDEXT1 : activate exit routine in SEXEC member EC9991
- MDG_UPDEXT2 : activate exit routine in SEXEC member EC9992
- MDG_UPDEXT3 : activate exit routine in SEXEC member EC9993
- MDG_UPDEXT4 : activate exit routine in SEXEC member EC9994
- MDG_UPDRETEXT : activate exit routine in SEXEC member EC9940
- MDG_UPD_CLEANUPEXT : activate exit routine in SEXEC member EC9949
- MDG_FILEXT1 : activate exit routine in SEXEC member EC9995
- MDG_FILEXT2 : activate exit routine in SEXEC member EC9996
- MDG_FILEXT3 : activate exit routine in SEXEC member EC9997
- MDG_STEXT : activate exit routine in SEXEC member EC9998
- MDG_APROT : activate exit routine in SEXEC member EC9999
- MDG_CXEXT : activate exit routine in SEXEC member EC9900
- MDG_PACTEXT : activate exit routine in SEXEC member EC9910
- MDG_PROUTEXT : activate exit routine in SEXEC member EC9920

The variables described in this section have these common features:

Variable Length:	1
Valid Values:	Y or N
Default Value:	N (except as noted below)
Location:	general profile

Note:

The default value of the global variable MDG_APROT is Y because it activates an ASG-supplied exit routine that protects the members of a project.

The default value of MDG_UPDEXT1 is set to Y to support the automatic naming of relationship members, based on the date and time of their creation. You should tailor SEXEC EC9991 if you want to define an alternative naming standard for relationship members.

Customizing Return From Buffers

To customize the action taken on return from an Assisted Update buffer, an Update buffer or an Edit buffer, use the following global variables.

- MDG_RETURN_AUPD : return from an Assisted Update buffer
- MDG_RETURN_UPD : return from an Update buffer
- MDG_RETURN_EDIT : return from an Edit buffer

These global variables determine the command that is executed when you press the appropriate key (PF3 by default) to return from an Assisted Update, an Update or an Edit buffer. By default a FILE command is issued from Assisted Update and a QUIT command from Update and Edit buffers.

By setting the values of the three variables, you may specify for each type of buffer whether the FILE or the QUIT command is issued when you press the PF key to return from the buffer.

The variables described in this section have the following common features:

Variable Length:	4
Valid Values:	FILE or QUIT
Default Value:	QUIT (except as noted below)
Location:	EC1060

Note:

The default value of MDG_RETURN_AUPD is FILE.

If the appropriate variable has not been declared or is null, default processing as described above will occur.

The global variable MDG_RETURN_EDIT is ignored on return from the Edit buffers named EXPERT, DRUCKJOB, PRINTJOB, and CMRBATCH.

Customizing Life Cycle Services (LCS)

Use the following global variables if you have tailored an ASG-supplied Life Cycle Model or if you have defined your own Life Cycle Model.

The phases, activities, subactivities, prerequisites, and deliverables of a ASG-supplied Life Cycle Model are defined in LIFE-CYCLE, PHASE, ACTIVITY, and LIFE-CYCLE-OBJECT-TYPE members.

Components of the panel interface—Life Cycle Services—which guide users through their project are generated from those members.

If you have used other member types or tailored ASG-supplied member types to define your Life Cycle Model, you need to specify the names of member types, clauses and relationships in the following global variables.

These global variables are used by the VXC and VX/VXA commands that check and generate the defined Life Cycle Model to enable Life Cycle Services (LCS).

Note:

Once you have changed the values of the global variables you must issue a VXA command to activate the new setting.

For details of ASG-supplied member types defining a Life Cycle Model, refer to *ASG-MethodManager Dictionary/Repository Information Model*.

Customizing Member Types Relevant for Life Cycle Services

Use the following global variables to specify the names of member types that define a user-defined Life Cycle Model, if the model has been defined in member types other than LIFE-CYCLE, PHASE, ACTIVITY and LIFE-CYCLE-OBJECT-TYPE. Specify the names of your member types in the relevant global variables. The names have to be specified as defined in the ENCODE-KEYWORDS clause of their MEMBER-TYPE member.

Member Type Defining a Life Cycle Model: MDG_VMODELL

Use MDG_VMODELL to specify the ENCODE-KEYWORDS entry of the MEMBER-TYPE member defining your life cycle model.

Variable Name:	MDG_VMODELL
Variable Length:	32
Valid Values:	ENCODE-KEYWORDS entry
Default Value:	LIFE-CYCLE
Location:	general profile

Member Type Defining a Phase: MDG_PHASE

Use MDG_PHASE to specify the ENCODE-KEYWORDS entry of the MEMBER-TYPE member defining a phase.

Variable Name: MDG_PHASE

Variable Length: 32

Valid Values: ENCODE-KEYWORDS entry

Default Value: PHASE

Location: general profile

Member Type Defining an Activity: MDG_AKTIV

Use MDG_AKTIV to specify the ENCODE-KEYWORDS entry of the MEMBER-TYPE member defining an activity.

Variable Name: MDG_AKTIV

Variable Length: 32

Valid Values: ENCODE-KEYWORDS entry

Default Value: ACTIVITY

Location: general profile

Member Type Defining a Life Cycle Object Type: MDG_ERGTYP

Use MDG_ERGTYP to specify the ENCODE-KEYWORDS entry of the MEMBER-TYPE member defining a life cycle object type.

Variable Name: MDG_ERGTYP

Variable Length: 32

Valid Values: ENCODE-KEYWORDS entry

Default Value: LC-TYPE

Location: general profile

Member Type Defining a Project: MDG_PROJEKT

Use MDG_PROJEKT to specify the ENCODE-KEYWORDS entry of the MEMBER-TYPE member defining a project.

Variable Name:	MDG_PROJEKT
Variable Length:	32
Valid Values:	ENCODE-KEYWORDS entry
Default Value:	PROJECT
Location:	general profile

Member Type Defining a Project View: MDG_PRJVIEW

Use MDG_PRJVIEW to specify the ENCODE-KEYWORDS entry of the MEMBER-TYPE member defining a project view.

Variable Name:	MDG_PRJVIEW
Variable Length:	32
Valid Values:	ENCODE-KEYWORDS entry
Default Value:	PROJECT-VIEW
Location:	general profile

Member Type Defining a Task: MDG_AUFTRAG

Use MDG_AUFTRAG to specify the ENCODE-KEYWORDS entry of the MEMBER-TYPE member defining a task.

Variable Name:	MDG_AUFTRAG
Variable Length:	32
Valid Values:	ENCODE-KEYWORDS entry
Default Value:	TASK
Location:	general profile

Member Type Defining a Document: MDG_DOKUMENT

Use MDG_DOKUMENT to specify the ENCODE-KEYWORDS entry of the MEMBER-TYPE member defining a document. This member type contains your project documentation.

Variable Name:	MDG_DOKUMENT
Variable Length:	32
Valid Values:	ENCODE-KEYWORDS entry
Default Value:	DOCUMENT
Location:	general profile

Customizing Clauses of Member Types Relevant for LifeCycle SERVICES

Use the following global variables to specify the names of clauses contained in member types that define a user-defined or a tailored ASG-supplied Life Cycle Model. Specify the names of the clauses in the relevant global variables. The names have to be specified as defined in the IDENTIFIED-BY clause of their ATTRIBUTE-TYPE member.

Clause Defining Help for Life Cycle Model, Phase, Activity, Subactivity: MDG_NOTE

Use MDG_NOTE to specify the name of the clause defining help for life cycle models, phases, activities, and subactivities.

By default help is called by entering the H line command in LCS. The clause itself must be a valid clause of a member type defining a life cycle model, a phase, an activity, and a subactivity.

Variable Name:	MDG_NOTE
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	HELP
Location:	general profile

Clause Defining Mandatory or Optional Results: MDG_TYPE

Use MDG_TYPE to specify the name of the clause that indicates if a result to be created is mandatory or optional.

The clause itself must be a valid clause of a member type defining an activity, a subactivity, and a life-cycle-object-type.

Variable Name:	MDG_TYPE
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	TYPE
Location:	general profile

Clause Containing Short Description of a Tool: MDG_TOOL

Use MDG_TOOL to specify the name of the clause that contains the short description of a tool to create or process a result.

The clause itself must be a valid clause of a member type defining a life-cycle-object-type.

Variable Name:	MDG_TOOL
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	TOOL
Location:	general profile

Clause Containing Short Description of Project Management Component: MDG_SDISP

Use MDG_SDISP to specify the name of the clause that contains the short description of a project management component.

The clause itself must be a valid clause of a member type defining a life cycle model, a phase, an activity, a subactivity, and a life-cycle-object-type.

Variable Name:	MDG_SDISP
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	OPTION
Location:	general profile

Clause Containing Long Description of Project Management Component: MDG_LDISP

Use MDG_LDISP to specify the name of the clause that contains the long description of a project management component.

The clause itself must be a valid clause of a member type defining a life cycle model, a phase, an activity, a subactivity, and a life-cycle-object-type.

Variable Name:	MDG_LDISP
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	OPTION-TEXT
Location:	general profile

Clause Defining Help for a Life-Cycle-Object-Type: MDG_DKHELP

Use MDG_DKHELP to specify the name of a clause defining help for a life-cycle-object-type.

By default help is called by entering the H line command in LCS.

The clause itself must be a valid clause of a member type defining a life-cycle-object-type.

Variable Name:	MDG_DKHELP
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	HELP
Location:	general profile

Clause Containing a Predefined Layout for a Document: MDG_DKLAYOUT

Use MDG_DKLAYOUT to specify the name of the clause that contains a predefined layout which can be copied to a document.

The clause itself must be a valid clause of a member type defining a life-cycle-object-type.

Variable Name:	MDG_DKLAYOUT
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	TEMPLATE
Location:	general profile

Clause Defining Execution of Tasks: MDG_COMMAND

Use MDG_COMMAND to specify the name of the clause that contains Manager Products commands, directives, macros etc. which call a tool or generate a result.

The clause itself must be a valid clause of a member type defining a life-cycle-object-type.

Variable Name:	MDG_COMMAND
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	COMMAND
Location:	general profile

Clause Defining Where a Result is to be Stored: MDG_ABLAGE

Use MDG_ABLAGE to specify the name of the clause that defines where a result is to be stored.

The clause itself must be a valid clause of a member type defining a life-cycle-object-type.

Variable Name:	MDG_ABLAGE
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	DATA-STORE
Location:	general profile

Clause Containing the Name of an Employee: MDG_VMUSER

Use MDG_VMUSER to specify the name of the clause that contains the name of an employee who is responsible for the execution of a task.

The clause itself must be a valid clause of a member type defining a task.

Variable Name:	MDG_VMUSER
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	EXECUTANT
Location:	general profile

Clause Defining the Current Status of a Member: MDG_COMSTATE

Use MDG_COMSTATE to specify the name of the clause that describes the quality or current status of the member's definition.

The clause itself must be a valid clause of a member type defining a life cycle model, a phase, an activity, a subactivity, and a life-cycle-object-type.

Variable Name:	MDG_COMSTATE
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	STANDING
Location:	general profile

Customizing Relationships Between Member Types Relevant for LifeCycle Services

Use the following global variables to specify the names of relationships between member types that define a user-defined or a tailored ASG-supplied Life Cycle Model. The names have to be specified as defined in the IDENTIFIED-BY clause of their ATTRIBUTE-TYPE member.

Relationship Between Life Cycle Model and Project: MDG_PJVM

Use MDG_PJVM to specify the name of the relationship between the member type defining a life cycle model and the member type defining a project.

The relationship itself must be a valid clause of a member type defining a life cycle model.

Variable Name:	MDG_PJVM
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	SEE
Location:	general profile

Relationship Between Life Cycle Model and Phase: MDG_VMPH

Use MDG_VMPH to specify the name of the relationship between the member type defining a life cycle model and the member type defining a phase.

The relationship itself must be a valid clause of a member type defining a life cycle model.

Variable Name:	MDG_VMPH
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	CONTAINS
Location:	general profile

Relationship Between Phase and Activity: MDG_PHAK

Use MDG_PHAK to specify the name of the relationship between the member type defining a phase and the member type defining an activity.

The relationship itself must be a valid clause of a member type defining a phase and an activity.

Variable Name:	MDG_PHAK
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	CONTAINS
Location:	general profile

Relationship Between Activity and Preceeding Life-Cycle-Object-Type: MDG_AKIN

Use MDG_AKIN to specify the name of the relationship between the member type defining an activity and the member type defining a life-cycle-object-type. The life-cycle-object-type is used as prerequisite for an activity.

The relationship itself must be a valid clause of a member type defining an activity or a subactivity.

Variable Name:	MDG_AKIN
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	INPUTS
Location:	general profile

Relationship Between Activity and Succeeding Life-Cycle-Object-Type: MDG_AKOUT

Use MDG_AKOUT to specify the name of the relationship between the member type defining an activity and the member type defining a life-cycle-object-type. The life-cycle-object-type is used as deliverable for an activity.

The relationship itself must be a valid clause of a member type defining an activity or a subactivity.

Variable Name:	MDG_AKOUT
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	OUTPUTS
Location:	general profile

Customizing Panels Used Under Life Cycle Services

To customize the layout of panels used under Life Cycle Services (LCS) use the following global variables.

Offset of the Long Description on a Panel: MDG_LDISPOFF

Use MDG_LDISPOFF to specify the offset of the long description on a panel under LCS.

If you are using ASG-supplied member types for defining a life cycle model the long description is generated from the OPTION-TEXT clause of the members defining the model. Otherwise the long description is generated from the clause specified in the global variable MDG_LDISP.

Variable Name:	MDG_LDISPOFF
Variable Length:	2
Valid Values:	any one or two-digit integer
Default Value:	14
Location:	general profile

Maximum Length of the Long Description on a Panel: MDG_LDISPLEN

Use MDG_LDISPLEN to specify the maximum length of the long description on a panel under LCS.

Variable Name:	MDG_LDISPLEN
Variable Length:	2
Valid Values:	any one or two-digit integer
Default Value:	50
Location:	general profile

Offset of the Short Description on a Panel: MDG_SDISPOFF

Use MDG_SDISPOFF to specify the offset of the short description on a panel under LCS.

If you are using ASG-supplied member types for defining a life cycle model the short description is generated from the OPTION clause of the members defining the model. Otherwise the short description is generated from the clause specified in the global variable MDG_SDISP.

Variable Name: MDG_SDISPOFF

Variable Length: 2

Valid Values: any one or two-digit integer

Default Value: 5

Location: general profile

Maximum Length of the Short Description on a Panel: MDG_SDISPLEN

Use MDG_SDISPLEN to specify the maximum length of the short description on a panel under LCS.

Variable Name: MDG_SDISPLEN

Variable Length: 2

Valid Values: any one or two-digit integer

Default Value: 10

Location: general profile

Offset of the Indicator for Mandatory or Optional Results: MDG_TYPOFF

Use MDG_TYPOFF to specify the offset of the indicator for mandatory or optional results on a panel under LCS.

If you are using ASG-supplied member types for defining a life cycle model the indicator is generated from the TYPE clause of the members defining the model. Otherwise the indicator is generated from the clause specified in the global variable MDG_TYPE.

Variable Name: MDG_TYPOFF

Variable Length: 2

Valid Values: any one or two-digit integer

Default Value: 63

Location: general profile

Maximum Length of the Indicator for Mandatory or Optional Results on a Panel: MDG_TYPLEN

Use MDG_TYPLEN to specify the maximum length of the indicator for mandatory or optional results on a panel under LCS.

Variable Name:	MDG_TYPLEN
Variable Length:	2
Valid Values:	any one or two-digit integer
Default Value:	63
Location:	general profile

Control whether Panel Invokes the Panel Display Exit (EC0995): MDG_GEN_PANEL_EXIT

Use MDG_GEN_PANEL_EXIT to control whether or not the VX and VXA commands enable a panel to invoke the Panel Display Exit (EC0955) when the panel is displayed.

(See "Tailoring the Panel Display" on page 184 for a description of the Panel Display exit.)

Variable Name:	MDG_GEN_PANEL_EXIT
Variable Length:	1
Valid Values:	Y or N
Default Value:	N
Location:	EC1060

Customizing Project Management Functions of Life Cycle Services

To customize Life Cycle Services (LCS) Project Management Functions use the following global variables.

Clause Documenting the History of a Project: MDG_PROJECT_HISTORY

Use MDG_PROJECT_HISTORY to specify the name of the clause that contains the project history entries.

The clause itself must be a valid text clause of the member type defining a project.

Note:

Project history entries may inform you for instance, when a life cycle model was assigned to the project or when an employee was assigned to or excluded from the project.

Variable Name:	MDG_PROJECT_HISTORY
Variable Length:	32
Valid Values:	HISTORY, ADMINISTRATIVE-DATA, COMMENT,DESCRIPTION, NOTE QUERY, or a user-defined clause of the type text
Default Value:	HISTORY
Location:	general profile

Automatic Update of Project History: MDG_PROJ_HIST_SWITCH

Use MDG_PROJ_HIST_SWITCH to activate or inactivate the automatic update of project history. Valid values:

- YES: automatic update of project history active
- NO: automatic update of project history inactive

Note:

If the automatic update is set to Yes, the global variable MDG_PROJECT_HISTORY must be set in conjunction to the name of a valid text clause of the member type defining a project

Variable Name:	MDG_PROJ_HIST_SWITCH
Variable Length:	3
Valid Values:	YES or NO
Default Value:	YES
Location:	general profile

Integration of New Members in the Project View of Current Project: MDG_PVIEW

Use MDG_PVIEW to specify whether a new member is to be included in the project view of the current project. Valid values are:

- Y: new member included in project view of the current project
- Any other value or none: new member not included in project view of the current project

Variable Name: MDG_PVIEW

Variable Length: 1

Valid Values: Y, any other value or none

Default Value: Y

Location: general profile

Enable/Disable Display of Prerequisites for an Activity: MDG_LCOT_INPUTS

Use MDG_LCOT_INPUTS to enable or disable the display of prerequisites for an activity. By default first the prerequisites and then the deliverables are displayed for a selected activity. Sometimes the list exceeds one screen page so that the user has to scroll down to see the deliverables. To suppress the prerequisites so that only the deliverables are displayed at the top of the list, use MDG_LCOT_INPUTS. Valid values:

- YES: prerequisites and deliverables displayed for an activity
- NO: only deliverables displayed for an activity

Variable Name: MDG_LCOT_INPUTS

Variable Length: 3

Valid Values: YES or NO

Default Value: YES

Location: general profile

Control Impact Analysis Output: MDG_PROJECTVIEW_RUCOUNTS

Use MDG_PROJECTVIEW_RUCOUNTS to determine whether PROJECT-VIEW member-type information is kept hidden from the user's view of the output from the MMR Impact Analysis functions, such as the USA and REFA commands.

Valid settings:

- Y: the output from REFA and USA commands ignores PROJECT-VIEW member-type information
- N: the output from REFA and USA commands counts occurrences of PROJECT-VIEW member-types

Variable Name:	MDG_PROJECTVIEW_RUCOUNTS
Variable Length:	1
Valid Values:	Y or N
Default Value:	N
Location:	general profile

Customizing Clauses Defining the Duration of a Task or a Project

Use the following global variables to specify the names of clauses that can be used in member types defining a task or a project.

The clauses define the:

- Planned beginning/end of a task or project
- Actual beginning/end of a task or project
- Estimated and actual duration of a task or project

Specify the names of the clauses in the relevant global variables, as defined in the IDENTIFIED-BY clause of their ATTRIBUTE-TYPE member.

Clause Defining the Planned Beginning: MDG_PLANBEGIN

Use MDG_PLANBEGIN to specify the name of the clause defining the planned beginning of a task or a project.

Variable Name:	MDG_PLANBEGIN
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	PLANNED-BEGIN
Location:	general profile

Clause Defining the Planned End: MDG_PLANEND

Use MDG_PLANEND to specify the name of the clause defining the planned end of a task or a project.

Variable Name:	MDG_PLANEND
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	PLANNED-END
Location:	general profile

Clause Defining the Actual Beginning: MDG_ACTBEGIN

Use MDG_ACTBEGIN to specify the name of the clause defining the actual beginning of a task or a project.

Variable Name:	MDG_ACTBEGIN
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	ACTUAL-BEGIN
Location:	general profile

Clause Defining the Actual End: MDG_ACTEND

Use MDG_ACTEND to specify the name of the clause defining the actual end of a task or a project.

Variable Name:	MDG_ACTEND
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	ACTUAL-END
Location:	general profile

Clause Defining the Estimated Duration: MDG_ESTDURATION

Use MDG_ESTDURATION to specify the name of the clause defining the estimated duration of a task or a project.

Variable Name:	MDG_ESTDURATION
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	ESTIMATED-DURATION
Location:	general profile

Clause Defining the Actual Duration: MDG_ACTDURATION

Use MDG_ACTDURATION to specify the name of the clause defining the actual duration of a task or a project.

Variable Name:	MDG_ACTDURATION
Variable Length:	32
Valid Values:	IDENTIFIED-BY entry
Default Value:	ACTUAL-DURATION
Location:	general profile

Activating User Exits for Life Cycle Services

To activate user exits for Life Cycle Services (LCS) use the following global variables. For details of user exits, refer to Chapter 7, "User Exits," on page 171.

- MDG_CHKMODEL : activate exit routine in SEXEC EC9960
- MDG_PROJDEF : activate exit routine in SEXEC EC9970
- MDG_PJLCASS : activate exit routine in SEXEC EC9971
- MDG_EMPLASS : activate exit routine in SEXEC EC9972
- MDG_PROJSEL : activate exit routine in SEXEC EC9973
- MDG_TASKSEL : activate exit routine in SEXEC EC9974
- MDG_VXEXT : activate exit routine in SEXEC EC9901

The variables described in this section have these common features:

Variable Length:	1
Valid Values:	Y or N
Default Value:	N
Location:	general profile

Customizing - Miscellaneous

Use the global variables described in this section to customize various areas.

Bypass Protection/Project View Insertion in RELABEL: MDG_RELABEL_BYPASS

Use MDG_RELABEL_BYPASS to specify whether member protection and project-view insertion are to be bypassed by the RELABEL command. Valid values:

- Y: Member protection/project-view insertion are bypassed
- N: Member protection/project view insertion are not bypassed (previous and current default RELABEL processing)

By default, if MDG_APROT is set to Y, RELABEL invokes the Protect Exit (EC9999) on the relabelled member, which, in LifeCycle Services, protects the member if there is an active project. In ToolSet Services, however, the Protect Exit only protects the relabelled member if it has been tailored specifically to protect members under ToolSet Services.

In addition, in LifeCycle Services, if MDG_PVIEW is set to Y, RELABEL inserts the relabelled member into the project view for the active project if there is one and protects it.

Setting MDG_RELABEL_BYPASS to Y bypasses all of the processing described above.

Variable Name:	MDG_RELABEL_BYPASS
Variable Length:	1
Valid Values:	Y or N
Default Value:	N
Location:	General Profile

Name of Administration RIM: MDG_ADMIN_UDS

Use MDG_ADMIN_UDS to specify the name of your Administration UDS Table if it is not DU777.

Variable Name:	MDG_ADMIN_UDS
Variable Length:	5
Valid Values:	Any valid UDS-Table name
Default Value:	DU777
Location:	EC1060

7 User Exits

This chapter includes these sections:

Global Exit Routines 172
Tailoring the Naming Convention Process 174
Tailoring the Assisted Update Process 175
Tailoring the File Process 178
Tailoring the Display of Relationships Between Members 180
Tailoring Member Protection 181
Tailoring CX Processing 182
Tailoring Panel Processing 182
Tailoring the Panel Display 184

Life Cycle Services 185
Tailoring Panel Display within Life Cycle Model 185
Tailoring Project Definition 186
Tailoring Assignment of Life Cycle to Project 187
Tailoring Assignment of User to Project 187
Tailoring Project Selection 187
Tailoring Task Selection 188
Tailoring VX/VXA Processing 188

Local Exit Routines 189
Tailoring the Naming Convention Process 189
Tailoring the Assisted Update Process 191
Tailoring the File Process 192

Dynamic Exit Routines 193
Tailoring the Return to the Panel Interface 193

User exits are set points in ASG-supplied software at which you can call *exit routines* (user-defined routines which allow you to tailor run control). The process of calling these routines is known as *taking user exits*. You will need to take user exits if the functionality of the standard routines does not meet the specific requirements of your organization.

Two types of exit routines are provided:

Global exit routines. These affect the entire environment, and must be coded in reserved SEXEC members. After coding, the routine is called via a variable.

Local exit routines. These are specific to certain member types and clauses.

Both global and local exit routines have to be defined in SEXEC members. An SEXEC is a special form of EXECUTIVE-ROUTINE in which macros can be specified. SEXECs are constructed onto the MP-AID as EXECUTIVE members with a CX command.

Refer to Chapter 9, "Member Types," on page 215 for details of the SEXEC member type.

Refer to Chapter 4, "Defining the Panel Interface," on page 45 for details of the CX command.

Coding examples for some exit routines are provided in the Administration Repository. The actual code depends on your specific corporate requirements, so there are not examples for each individual exit routine.

To code exit routines, you need a fundamental knowledge of Manager Products commands, directives and functions.

Note:

You should not use user exits to output data to the screen, because this interrupts the normal flow of processing and can cause erroneous results.

Global Exit Routines

Global exit routines are defined in SEXEC members with the naming convention EC####. Internally, the routines are called using the MP-AID name of these members. Hence:

- The name must not be changed
- The routines must be defined in the reserved SEXEC

There are sixteen global exit routines available to tailor run control. Each global exit routine is associated with a specific user exit, and can tailor a specific part of ASG software.

To take a specific user exit (to call a specific global exit routine), set an appropriate global variable in the GROUP member GR-MMR-EXITS to Y. The table below shows how to call each global exit routine:

To Tailor	Use Global Exit Routine	Called via Variable
Naming convention process	EC9980	MDG_NAMEXT1
	EC9981	MDG_NAMEXT2
Assisted update process	EC9991	MDG_UPDEXT1
	EC9992	MDG_UPDEXT2
	EC9993	MDG_UPDEXT3
	EC9994	MDG_UPDEXT4
	EC9940	MDG_UPDRETEXT
	EC9949	MDG_UPD_CLEANUPEXT
File process	EC9995	MDG_FILEXT1
	EC9996	MDG_FILEXT2
	EC9997	MDG_FILEXT3
Display of relationships between members	EC9998	MDG_STEXT
Member protection	EC9999	MDG_APROT
CX process	EC9900	MDG_CXEXT
Panel interface process	EC9910	MDG_PACTEXT
	EC9920	MDG_PROUTEXT
Display of panels	EC0955	N/A **Note:** This exit is only invoked by panels that have been generated with MDG_GEN_PANEL_EXIT = Y (see "Customizing the Panel Interface" on page 123).

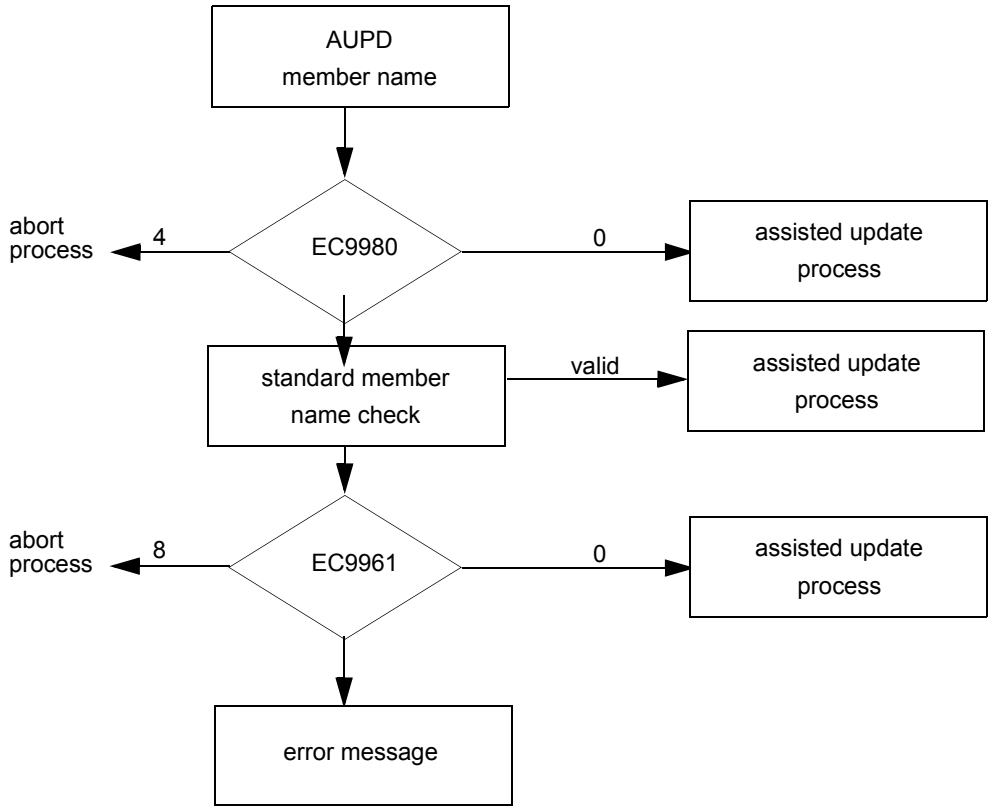
The global exit routines are described in detail in the following sections.

Tailoring the Naming Convention Process

MethodManager requires strict adherence to standard naming conventions, which are automatically checked before a member is created or updated. To use members with non-standard names, you must define your own naming conventions, using global exit routines in SEXEC members EC9980 and EC9981.

Figure 53 shows at which point the two global exit routines can be called in the naming convention process:

Figure 53 • Tailoring the Naming Convention Process



Both EC9980 and EC9981 receive the member name in the parameter &PO.

In EC9980, the following have to be set for further processing:

- Return Code 0 (EXIT 0). The member name has been validated by the standard naming convention check. These command variables must be set for further processing:
  - MDC_NAMING_ALIAS. Receives the two digit member type alias (defined in the ALIAS clause of the MEMBER-TYPE or RELATIONSHIP-TYPE definition)
  - MDC_NAMING_MTYPE. Receives the member type identifier keyword (defined in the ENCODE-KEYWORDS or LONG-NAME clauses of the MEMBER-TYPE, or the PRIMARY-NAME clause of the RELATIONSHIP-TYPE)
- Return-Code 4 (EXIT 4). The member name does not correspond to a valid naming convention. Stop further processing. Output error message DM41670E.
- Return-Code 8 (EXIT 8). No checks done by EC9980. Processing continues with the standard naming convention check.

Note:

EC9980 contains a coding example.

EC9981 is called if the standard naming convention check fails. In EC9981 the following return codes have to be set for further processing:

- Return Code 0 (EXIT 0). The command variables MDC_NAMING_ALIAS and MDC_NAMING_MTYPE (described above) have to be set for further processing:
- Return-Code 4 (EXIT 4). The member name does not correspond to a valid naming convention. Abort further processing. Output error message DM41670E.
- Return-Code 8 (EXIT 8). No further processing.

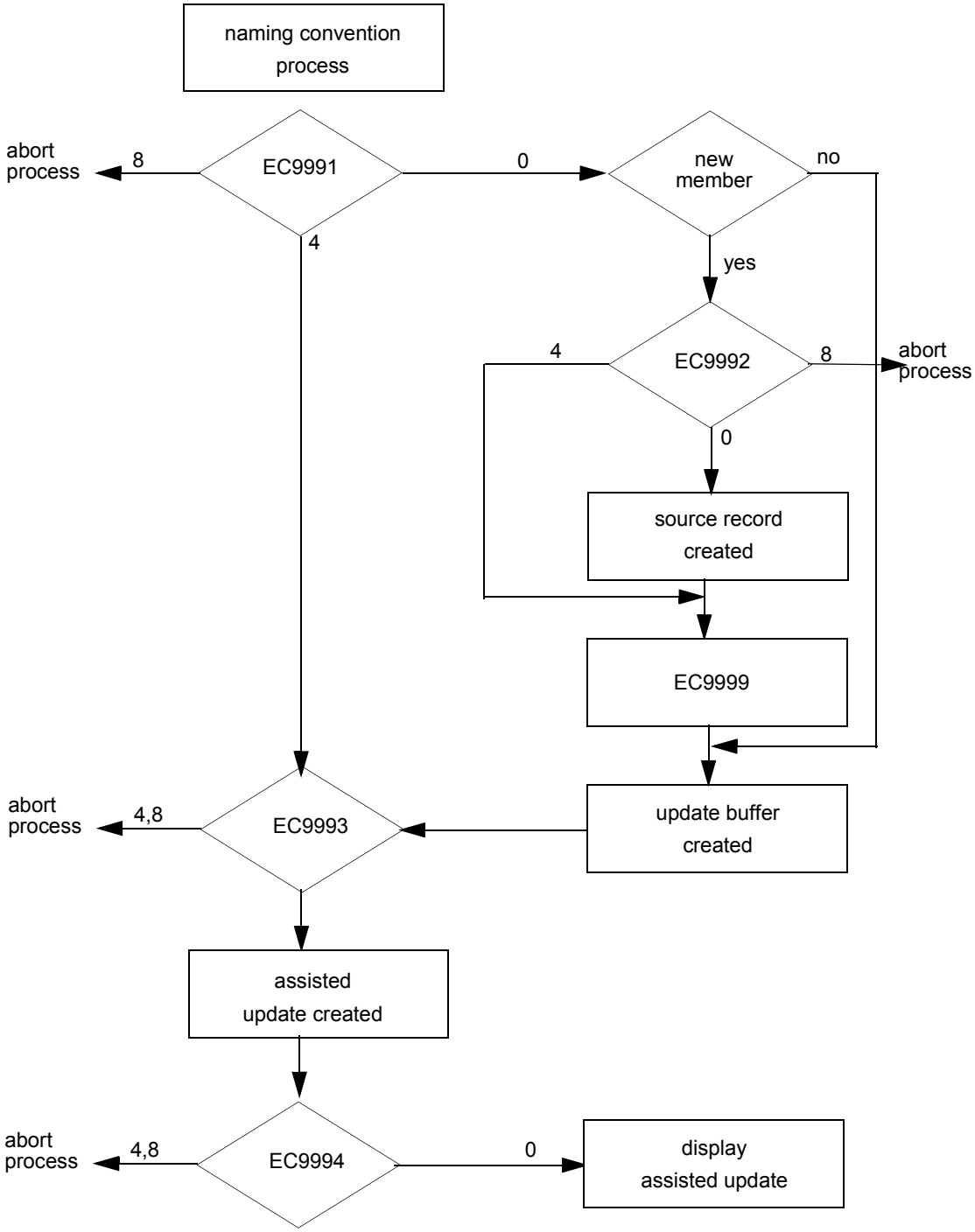
Tailoring the Assisted Update Process

The following global exit routines tailor the processing of members during assisted updates: EC9991, EC9992, EC9993, and EC9994. The routines all receive the following information:

- The member name in the parameter &P0
- The member type in the parameter &P1
- The member type alias in the parameter &P2

Figure 54 shows at which point in the assisted update process the global exit routines can be called:

Figure 54 • Tailoring the Assisted Update Process



EC9991 controls the processing of existing members and supports the automatic naming of relationship members. For example, you might use EC9991 to check if certain users have the required authority to access repository members. In EC9991 the following must be set for further processing:

- Return Code 0 (EXIT 0). Processing continues
- Return Code 4 (EXIT 4). Update buffer opened. Processing continues
- Return Code 8 (EXIT 8). Stop further processing

EC9992 controls the processing of new members. For example, you might use EC9992 to check that users have the authority to create new members in the repository. In EC9992 the following must be set for further processing:

- Return Code 0 (EXIT 0). Processing continues

  Note:

  If processing ends with return code 0, this acts as if EC9992 had not been called. Coding that results in return code 0 is ignored when the member's source record is created.

- Return Code 4 (EXIT 4). A new source only member has been created. Processing continues
- Return Code 8 (EXIT 8). Stop further processing

EC9993 can be used (for example) to insert administrative data in a clause of the member definition. In EC9993 the following must be set for further processing:

- Return Code 0 (EXIT 0). Processing continues
- Return Code 4 (EXIT 4). No further processing. The update buffer remains open
- Return Code 8 (EXIT 8). No further processing. The update buffer is closed

EC9994 can be used (for example) to place a certain clause at the top of the assisted update. In EC9994 the following must be set for further processing:

- Return Code 0 (EXIT 0). Processing continues
- Return Code 4 (EXIT 4). No further processing. The update buffer is closed
- Return Code 8 (EXIT 8). No further processing. The update buffer is closed

Note:

EC9994 contains a coding example.

The return from assisted update to the panel interface can be tailored using the global exit routines EC9940 and EC9949.

EC9940 is invoked immediately a command is issued or PF key pressed to close the assisted update buffer. EC9940 receives no parameters.

The following must be set for further processing:

- Return Code 0 (EXIT 0). Processing continues
- Return Code 4 (EXIT 4). No further processing. The update buffer remains open
- Return Code 8 (EXIT 8). No further processing. The update buffer remains open

EC9949 is invoked on return from an assisted update buffer after any file or cancel processing has been completed. The parameters passed to the exit reflect the result of the file/cancel processing and are:

- In &P0: name of member in assisted update buffer
- In &P1: one of these keywords:

ENCODED	(member has been successfully encoded)
UNCHANGED	(member definition has not changed)
CANCELLED	(update of member has been cancelled)
ACCEPTED	(member definition has been saved but not encoded)
or RC = n	(filing of member has resulted in encode errors or has been aborted by request of other exits; n represented the return code causing the termination)

In all circumstances except where &P1 is RC = n, the assisted update buffer will normally already be closed when the exit is invoked, unless another user exit has caused it to remain open. Where &P1 is RC = n, the assisted update buffer will normally remain open when the exit is invoked, unless another user exit has caused it to be closed.

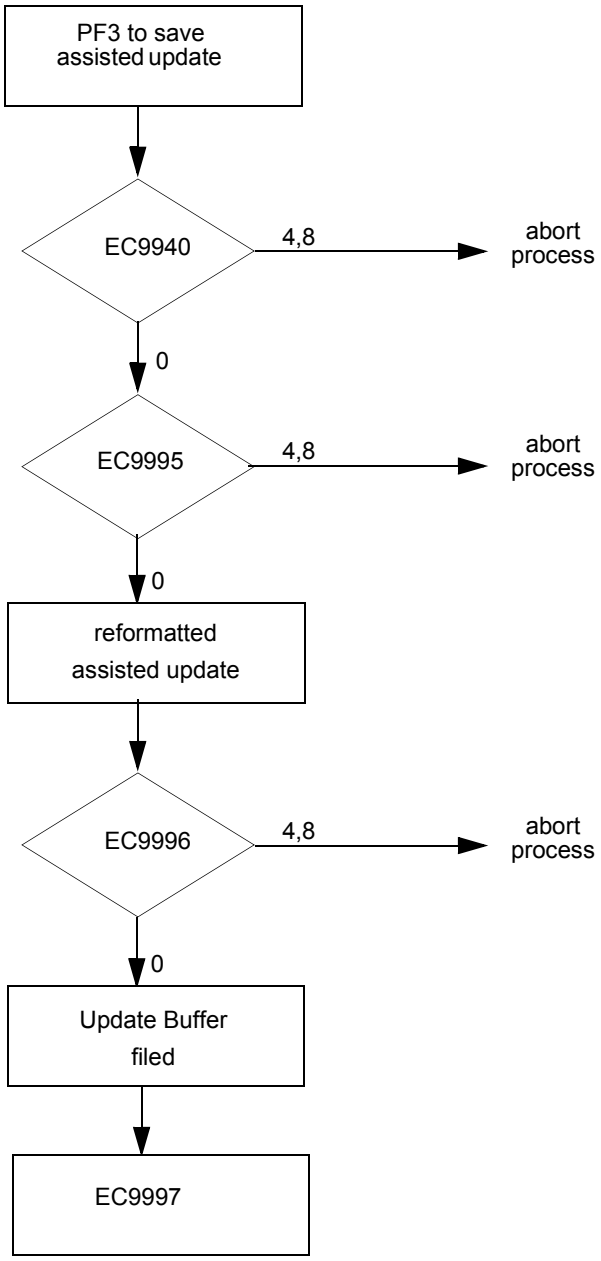
Tailoring the File Process

A member must be reformatted for encoding in the repository after it has been created or updated in an assisted update. This is normally done automatically.

If necessary, you can tailor the file process using the global exit routines EC9995, EC9996, and EC9997.

The following flowchart shows at which point each routine can be called:

Figure 55 • Global Exit Routines for the File Process



Note:

See "Tailoring the Assisted Update Process" on page 175 for a description of EC9940.

EC9995 can be used (for example) to check that valid entries are contained in the CATALOG clause of a member. EC9995 receives the member name in the parameter &P0. In EC9995, the following return codes have to be set for further processing:

- Return Code 0 (EXIT 0). Processing continues with the standard routine.
- Return Code 4 (EXIT 4). No further processing. The update buffer remains open.
- Return Code 8 (EXIT 8). No further processing. The update buffer is closed.

Note:

EC9995 contains a coding example.

EC9996 can be used (for example) to insert administrative data automatically in a clause of the member definition. EC9996 receives the member name in the parameter &PO. In EC9996, the following return codes have to be set for further processing:

- Return Code 0 (EXIT 0). Processing continues with the standard routine.
- Return Code 4 (EXIT 4). No further processing. The update buffer remains open.
- Return Code 8 (EXIT 8): No further processing. The update buffer is closed.

EC9997 can be used (for example) to check if dummies have been created by references from the encoded member. EC9997 receives the following information in the following parameters:

- The member name in &P0
- The condition code of the FILE command (&CCOD) in &P1.
- The message number of the FILE command (&MSNO) in &P2.
- The severity code of the FILE command (&MSLV) in &P3.

If return codes are set in SEXEC EC9997 they will be ignored.

Tailoring the Display of Relationships Between Members

Relationships between members in the repository can be queried using the line commands REFA, REF (show references) and USA, US (show usages).

If the resulting display does not meet your specific requirements, use the exit routine EC9998 to tailor the output.

The exit routine receives this information in these parameters:

- The standard output line in &P0
- CONSTITUTES/USES in &P1
- In &P2 either:

  0 = relationships are further decomposed

  or

  9 = relationships have been decomposed already
- The member name in &P3
- The separator in &P4
- The level number in &P5
- The member type in &P6

If return codes are set in the SEXEC EC9998 they will be ignored.

Note:

EC9998 contains a coding example.

Tailoring Member Protection

In LifeCycle Services (LCS), members are by default protected by the project for which they have been created.

This protection function is defined in the exit routine EC9999 via a PROTECT command in which the project is specified as the OWNER of the new member.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of the PROTECT command.

You can also tailor EC9999 to protect new members in ToolSet Services.

Note:

The coding that ensures member protection in LifeCycle Services must not be changed when tailoring the global exit routine.

In LifeCycle Services, EC9999 is called, by default, after the source record of a new member has been created.

EC9999 receives the following information in these parameters:

- The member name in &P0
- The member type in &P1
- The member type alias in &P2

In EC9999, these return codes have to be set for further processing:

- Return Code 0 (EXIT 0). Member successfully protected.
- Return Code 4 (EXIT 4). Member not protected.
- Return Code 8 (EXIT 8). Member not protected.

Note:

EC9999 contains a coding example that shows how this routine can be utilized for ToolSet Services.

Tailoring CX Processing

The output from a CX command can be tailored using the CX exit (EC9900).

EC9900 is invoked at the end of CX processing before a member is written to the MP-AID.

The exit routine receives the following information in these parameters:

If the member CX-ed is a SEXEC or FMT-SCREEN:

- SEXEC or FMT-SCREEN in &P0
- The member name to be written to the MP-AID in &P1
- The name of the array containing the generated source in &P2

No return codes are checked.

Tailoring Panel Processing

Panel processing can be tailored using the Panel Action exit (EC9910) and the Panel Routing exit (EC9920).

The Panel Action exit (EC9910) is called whenever the Enter key or a PF key is pressed within the panel interface. If there is an action to be performed, EC9910 is invoked before the action is performed with &P0 = BEFORE and after the action is performed with &P0 = AFTER. If there is no action to be performed, EC9910 is invoked with the parameter NO-INPUT.

In addition, EC9910 receives these parameters:

- The current panel name in &P1
- The PF key used in &P2
- The command specified in &P3

If both &P2 and &P3 are null, the action is determined solely by the panel specified in &P1.

These return codes must be set in EC9910 for further processing:

If &P0 = BEFORE:

- Return Code 0 (EXIT 0). Proceed with action
- Return Code 4 (EXIT 4). No further processing
- Return Code 8 (EXIT 8). No further processing

If &P0 = AFTER:

- No return codes are checked.

If &P0 = NO-INPUT:

- Return code 0 (EXIT 0). No further processing
- Return code 4 (EXIT 4). Proceed with normal processing
- Return code 8 (EXIT 8). Proceed with normal processing

The Panel Routing exit (EC9920) is called whenever a menu option is invoked. It receives the following information in the following parameters:

- The name of the panel whose option is to be invoked in &P0
- The option in &P1

The following return code must be set in EC9920 for further processing:

- Return Code 0 (EXIT 0). Proceed with action
- Return Code 4 (EXIT 4). No further processing
- Return Code 8 (EXIT 8). No further processing

Tailoring the Panel Display

The data displayed on a panel may be tailored using the Panel Display exit (EC0995), if the panel has been generated with the variable MDG_GEN_PANEL_EXIT set to Y.

EC0995 is invoked before each line of the panel is displayed and allows each field on the line, its position, length and protection to be tailored.

The data available to the exit is held in the following command variables:

Variable	Contents
MDC_LINE_NUM	Current line number of panel
MDC_LV(*field-num*)	Field contents
MDC_LV_SP(*field-num*)	Field start position (column number)
MDC_LV_LEN (*field-num*)	Field length
MDC_LV_PROT (*field-num*)	Field protection (see FMT-SCREEN member definition for values)

where *field-num* is a number from 1 to 10 relating to a field reading from left to right across the line.

In addition, the position of the cursor on the panel can also be tailored by setting the command variables MDC_CURSOR_ROW and MDC_CURSOR_COLUMN to the desired row and column positions.

Note:

No attempt is made to ensure that fields tailored are valid. Consequently panel errors will occur if fields are assigned invalid values.

Use of the exit may have a detrimental effect on performance. The number of fields on the panel and the execution time of the exit both affect the time taken to display and manipulate the panel. (This is particularly noticeable in LIST panels.)

Life Cycle Services

There are seven global exit routines available to tailor run control within Life Cycle Services. Each global exit is associated with a specific part of Manager Products software.

To take a specific user exit (to call a specific global exit routine), set an appropriate global variable in the GROUP member GR-MMR-LCS-EXITS to Y. The table below shows how to call each global exit routine:

To Tailor	Use Global Exit Routine	Called via Variable
Panel Display within Life Cycle Model	EC9960	MDG_CHKMODEL
Project Definition	EC9970	MDG_PROJDEF
Life Cycle to Project Assignment	EC9971	MDG_PJLCASS
User to Project Assignment	EC9972	MDG_EMPLASS
Project Selection	EC9973	MDG_PROJSEL
Task Selection	EC9974	MDG_TASKSEL
VX/VXA Process	EC9901	MDG_VXEXT

Tailoring Panel Display within Life Cycle Model

The display of panels within a Life Cycle Model can be tailored using the global exit routine EC9960. This exit routine can be used both to allow or prohibit the selection of a certain Life Cycle member and to determine the data displayed on the resulting panel.

In ENTRY mode, which is initiated on the selection of a Life Cycle member, EC9960 determines whether or not the selection is permitted. In LIST mode, which is initiated before a panel displays, EC9960 determines the data to display on the panel.

EC9960 receives the following information in these parameters:

ENTRY MODE:

- In &P0: ENTRY (keyword)
- In &P1: member-name selected
- In &P2: member-type selected

Return codes:

- Return Code 0 (EXIT 0). Displays the following panel, that is, the next activity menu
- Return Code 4 (EXIT 4) or 8 (EXIT 8). Does not display the following panel

LIST MODE

- In &P0: LIST (keyword)
- In &P1: panel type (naming prefix of selected Life Cycle member)

LT/VM	Life Cycles
PH	Phases
AK/AC	Activities
SU	Sub-activities
LT/ET	Life Cycle Object Types

Return codes:

- Return Code 0 (EXIT 0). Displays the panel, together with information stored in MDC_EXMSG
- Return Code 4 (EXIT 4) or 8 (EXIT 8). Displays the panel, ignoring any information stored in MDC_EXMSG

See member EC9960 in your Administration dictionary for further information.

Tailoring Project Definition

The definition of a new project can be tailored using the global exit routine EC9970.

EC9970 receives the following information in these parameters:

- In &P0: project name
- In &P1: project long name

Return codes:

- Return Code 0 (EXIT 0). Project definition successfully inserted into Repository
- Return Code 4 (EXIT 4): Project to be defined by Life Cycle Services
- Return Code 8 (EXIT 8). Project not defined

Tailoring Assignment of Life Cycle to Project

The assignment of a Life Cycle model to a project can be tailored using the global exit routine EC9971.

EC9971 receives the following information in these parameters:

- In &P0: project name
- In &P1: Life Cycle name

Return codes:

- Return Code 0 (EXIT 0). Assignment performed. SEE clause of project refers to Life Cycle model
- Return Code 4 (EXIT 4): Life Cycle Services to maintain the SEE clause of the project
- Return Code 8 (EXIT 8). Life Cycle model not assigned to project

Tailoring Assignment of User to Project

The assignment of a user to a project can be tailored using the global exit routine EC9972. EC9972 must perform the SECURITY command necessary to allow the user access to the project.

EC9972 receives the following information in these parameters:

- In &P0: project name
- In &P1: user name

Return codes:

- Return Code 0 (EXIT 0). Assignment successful
- Return Code 4 (EXIT 4): Life Cycle Services to maintain security
- Return Code 8 (EXIT 8). User not assigned to project

Tailoring Project Selection

The selection of a project can be tailored using the global exit routine EC9973.

EC9973 receives the following information in these parameters:

- In &P0: project name

Return codes:

- Return Code 0 (EXIT 0). Project selection successful
- Return Code 4 (EXIT 4): Life Cycle Services to maintain project
- Return Code 8 (EXIT 8). Project not selected

Tailoring Task Selection

The selection of a task can be tailored using the global exit routine EC9974.

EC9974 receives the following information in these parameters:

- In &P0: task name

Return codes:

- Return Code 0 (EXIT 0). Task selection successful
- Return Code 4 (EXIT 4): Life Cycle Services to set the variables for the selected task
- Return Code 8 (EXIT 8). Task not selected, terminate further processing

Tailoring VX/VXA Processing

The output of a VA or VXA command can be tailored using the global exit routine EC9901. This exit is invoked at the end of VX/VXA processing, just before a member is written to the MP-AID.

EC9901 receives the following information in these parameters:

- In &P0: SCREEN/SCREEN-EXEC/LCOT (keyword)
- In &P1: the name of the member to be written to the MP-AID
- In &P2: the name of the array containing generated source
- In &P3: the name of an additional array containing generated source (when &P0 = LCOT only)

No return codes are checked.

Local Exit Routines

Local exit routines can be defined for particular member types and clauses. They are used to tailor the processing of only those member types and clauses for which they are defined.

Local exit routines need not be called by variables and have no predefined names.

Local exit routines must be:

- Defined in SEXEC members
- Constructed onto the MP-AID using the CX command

To avoid naming conflicts in future releases, ASG recommends defining local exit routines in SEXEC members with the naming convention EW####.

There are three types of local exit routine available, for tailoring the following processes:

- Naming conventions (see the example in the SEXEC EW6010)
- Assisted update (see the example in the SEXEC EW3510)
- File (see the example in the SEXEC EW3520)

These routines are explained in the following sections.

Each SEXEC member in which a local exit routine is defined must be constructed onto the MP-AID before it can be called.

Tailoring the Naming Convention Process

These local exit routines are specific to certain member types. Each routine can be used to check the naming conventions of members of one specified type.

You need to specify, in the NAMING-EXIT clause of the relevant MEMBER-TYPE or RELATIONSHIP-TYPE, the name of the SEXEC defining the local exit routine for that member type. You also need to generate the MEMBER-TYPE or RELATIONSHIP-TYPE.

If a local exit is taken for a member type, it replaces the standard naming convention check for that member type. If one or both of the global exit routines defined in SEXEC members EC9980 and EC9981 are called, the local exit is taken after EC9980 and before EC9981 (see Figure 53 on page 174).

The naming convention check of a member type can be executed either by:

- The standard check and the global exit routines in EC9980 or EC9981 (if they are called)
- A local exit routine and the global exit routines in EC9980 or EC9981 (if they are called)

Because these local exit routines not only control the naming conventions but also the name-related interrogation of members of a certain type, the information they receive depends on the required processing.

The local exit routines receive the following information when naming conventions are checked (for example, when a member is being updated):

- "NAME" in &P0
- The member name in &P1
- The member type in &P2
- The member type alias in &P3

The local exit routines receive the following information when members are listed according to their member type naming conventions (for example, when using the S line command to list all members of a selected type on a selection list panel):

- "INTERROGATE" in &P0
- The naming convention of the member type in &P1 (given in the NAMING clause in the MEMBER-TYPE or RELATIONSHIP-TYPE definition)
- The member type in &P2
- The member type alias in &P3

The following return codes have to be set in the SEXEC member defining the exit routine, if members are to be checked for the naming convention/s of their type (NAME):

- Return code 0. The name is valid for members of the checked type
- Return code 4. The name is invalid for members of the checked type
- Return code 8. The name is invalid for members of the checked type

The following return codes have to be set in the SEXEC member defining the exit routine, if members are to be listed according to the naming conventions of their type (INTERROGATE):

- Return code 0. The selection criteria is valid. The command variable MDC_NAMING_ALIAS has to be set to KEPT IN *name* for further processing, where *name* is the name of a KEPT-DATA list containing the selected members. The KEPT-DATA list has to be created in the exit routine.
- Return code 4. The selection criteria is valid. Processing continues with the standard routine. The standard routine automatically receives the naming convention of the member type in the command variable MDC_NAMING_CONV.
- Return code 8. The selection criteria is invalid.

Note:

EW6010 contains a coding example.

Screen output (initiated by directives such as SAY or WRITEL) must not be created in local exit routines of the above type, as this might cause unpredictable results.

Tailoring the Assisted Update Process

These local exit routines are specific to certain clauses. Each local exit routine can be used to tailor the display of one clause during assisted updates.

You need to specify, in the EDIT-EXEC-1 clause of the relevant ATTRIBUTE-TYPE, the name of the SEXEC member defining the local exit routine for a clause. You also need to generate the ATTRIBUTE-TYPE.

If a local exit is taken for a certain ATTRIBUTE-TYPE, it replaces the standard routine specified in the EDIT-CODE-1 clause of the ATTRIBUTE-TYPE member.

If a local exit is taken for the assisted update process, the local exit routine is called after global exit routines defined in the SEXEC members EC9991, EC9992, or EC9993 and before EC9994 (see Figure 54 on page 176).

The local exit routines receive the following information in the following parameters:

- In &P0: internal information (ASG only)
- In &P1: the number of the line that contains the first entry (the keyword) of the clause (assigned to the command variable MDC_AUPD_BUF)
- In &P2: the number of the line containing the last entry of the clause (assigned to the command variable MDC_AUPD_BUF)
- In &P3: the current line erase character

Note:

The command variable MDC_AUPD_BUF holds the member definition after it has been subjected to the AMEND process.

EW3510 contains a coding example.

Tailoring the File Process

These local exit routines are specific to certain clauses. They can be used to tailor the reformatting of clauses for encoding in the repository. For each clause of the RIM exactly one local exit routine can be defined to control the reformatting for encoding.

You need to specify, in the EDIT-EXEC-2 clause of the relevant ATTRIBUTE-TYPE, the name of the SEXEC member defining the local exit routine for a clause. You also need to generate the ATTRIBUTE-TYPE.

If a local exit is taken for an ATTRIBUTE-TYPE, it replaces the standard routine specified in EDIT-CODE-2 of the ATTRIBUTE-TYPE.

If global exit routines are called for the file process, the local exit will be taken after any global exit routines defined in SEXEC members EC9995, EC9996, or EC9997 (see Figure 55 on page 179).

The local exit routines receive the following information in the following parameters:

- In &P0: internal information (ASG only)
- In &P1: the number of the line containing the first entry (the keyword) of the clause (assigned to the command variable MDC_AUPD_BUF)
- In &P2: the number of the line containing the last entry of the clause (assigned to the command variable MDC_AUPD_BUF)
- In &P3: the current line erase character

Note:

EW3520 contains a coding example.

Dynamic Exit Routines

Dynamic exit routines are exit routines that are invoked once only. They are invoked by setting an appropriate global variable to the name of the exit routine to be executed. On execution of the exit routine, the global variable is reset to null. For this reason the exit routine is not invoked when the global variable is next tested.

The exit routine must be an SEXEC or EXECUTIVE-ROUTINE member which has been constructed on to the MP-AID.

Tailoring the Return to the Panel Interface

Return from any panel or BLT buffer to the previous panel or buffer may be controlled by the General Return Exit.

The purpose of the exit is to tailor a particular situation relating to a specific panel or buffer. The exit is activated when this particular situation arises. The global variable MDG_GRETEXT is set to the name of the exit routine to be invoked and the next return from the panel or buffer invokes the exit. When invoked the exit checks whether the environment remains the same as it was when the exit was activated (another panel or buffer may have opened subsequently, for example).

If the environment is the same, the exit performs its task and is automatically deactivated until the particular situation arises again.

If the environment is not the same, the exit must re-activate itself (that is, reset MDG_GRETEXT to the name of the exit routine) so that it will be invoked the next time the RETURN function is issued.

An example where such an exit could be useful is for a LIST screen where several member names are to be selected and stored as references in a new member after the RETURN function is invoked.

The exit routine receives the following information in these parameters:

- RETURN (save buffer, current buffer remains open) in &P0

or

- EXIT (exit buffer) in &P0

These return codes must be set by the exit routine for further processing:

- Return code 0 (EXIT 0): Continue standard processing
- Return code 4 (EXIT 4): No further processing
- Return code 8 (EXIT 8): No further processing.

8 Macros

This chapter includes these sections:

Macro Descriptions 198
:BROWSE 198
:FMTSCREEN 199
LPARM 200
NAMKO 201
NAMKOT 206
OUTE 207
RETAIN 208
VCHNG 209
VSEARCH 210
XFILE 211

A macro combines a set of instructions, frequently used in programming, under a unique name. Instead of several lines of code you just enter the macro name.

Benefits:

- Macros make coding faster and easier.
- Macros contribute to a more structured programming.
- Macros make code more readable.
- Macros are faster than calls to other Executive Routines.

Macros are defined in SEXEC members with the naming convention EM####. After the definition the macro has to be constructed onto the MP-AID, using the CX-command.

Macros can be used for coding:

- In the CONTENTS clause of SEXEC members
- In the COMMAND clause of LIFE-CYCLE-OBJECT-TYPE members

A macro, used in a member of the above mentioned types, is expanded into its instructions, when the member is constructed onto the MP-AID.

Figure 56 shows the CONTENTS clause of the SEXEC member EW7030 where the :RETAIN macro is specified to retain the Executive Routine in virtual storage.

Figure 56 • The Repository Member EW7030

```
CONTENTS
LITERAL #
:RETAIN CLASS=2
MPR #MPEAW7030#;
EXIT &CCOD
```

Figure 57 shows the constructed SEXEC EW7030 on the MP-AID. The RETAIN macro has been expanded into two lines of code.

Figure 57 • The MP-AID Member

```
MPXX LITERAL=#
IF MDG_TRC1 EQ 1 THEN TRACE RESULTS
GLOBAL MDG_RETAIN2
IF MDG_RETAIN2 = 1 THEN RETAIN
MPR #MPEAW7030#;
EXIT &CCOD
```

Note:

The colon (:) is the indicator for a macro. Therefore the colon (:) must not be used as literal in SEXEC or LIFE-CYCLE-OBJECT-TYPE members.

The MPXX and TRACE RESULTS directives in Figure 57 are inserted automatically in the EXECUTIVE member. This is done by the CX command for all SEXEC members when they are constructed onto the MP-AID.

There are more macros than documented in this chapter. Please note that the documented macros are reserved for the Systems Administrator. You may use other macros at your own risk, because ASG does not guarantee their compatibility in future releases of our software.

These macros are exclusively used in LIFE-CYCLE-OBJECT-TYPE members. They must *not* be used in SEXEC members:

- :CASE
- :DCSTANDARD
- :DISPLAY
- :LEVEL
- :LINE-COMMAND
- :STANDARD

For more information see "Macros" on page 348.

These macros can be used in SEXEC and LIFE-CYCLE-OBJECT-TYPE members:

- :BROWSE
- :FMTSCREEN
- :LPARM
- :NAMKO
- :NAMKOT
- :OUTE
- :RETAIN
- :VCHNG
- :VSEARCH
- :XFILE

These macros are documented in "Macro Descriptions" on page 198.

The :INCLUDE macro must only be used in SEXEC members. It is also documented in "Macro Descriptions" on page 198.

The following macros are still available to keep the Manager Products compatible with previous releases. Their use should be discontinued, because the functionality of these macros has been replaced by Manager Products procedures language directives:

- :DO FOR
- :DO FOREVER
- :DO WHEN
- :DO WHILE
- :ELSE
- :ENDDO
- :ENDIF
- :IF
- :LEAVE
- :LOOP

For details of these macros, refer Appendix B,"Superseded Macros," on page 401.

Macro Descriptions

This section describes the macros provided by Manager Products. The macros are documented in alphabetic order of macro name.

:BROWSE

The :BROWSE macro displays output in a new Lookaside buffer without deleting the contents of the previous buffer.

The :BROWSE macro can be used in SEXEC and LIFE-CYCLE-OBJECT-TYPE members.

In contrast to the primary commands BROWSE and LOOKASIDE, the :BROWSE macro used in an Executive Routine, that is running in batch mode, will be ignored but not abort the process.

For example, if you specify:

```
:BROWSE
SAY #TEST SUCCESSFULLY COMPLETED#
```

the message `TEST SUCCESSFULLY COMPLETED` displays in a Lookaside buffer from where you return to the previous buffer using PF3. Without the :BROWSE macro the contents of the previous buffer would be deleted.

Note:

The hash characters (#) are used as literals in the example.

:FMTSCREEN

The :FMTSCREEN macro calls a panel defined in a FMT-SCREEN member.

The :FMTSCREEN macro can be used in SEXEC and LIFE-CYCLE-OBJECT-TYPE members.

For example, if you specify:

```
:FMTSCREEN SC-TD22000
```

the current panel defined in the FMT-SCREEN member SC-TD22000 is set up refreshed again.

For example, if you specify:

```
LOCAL MDL_PANEL
SET MDL_PANEL MFSTD22000
:FMTSCREEN MDL_PANEL NEXT
```

the NEXT keyword indicates that the panel SC-TD22000 identified by its MP-AID name MFSTD22000 in the local variable MDL_PANEL is to be set up for display.

Note:

The global variables that contain the information required for setting up a panel must either be set in the SEXEC or LIFE-CYCLE-OBJECT-TYPE member which uses the :FMTSCREEN macro or in the panel called by the :FMTSCREEN macro. Use the CALLS exec AT INIT clause of the FMT-SCREEN member type if the global variables are to be set directly in the called panel.

For details of the FMT-SCREEN member type, refer to Chapter 9, "Member Types," on page 215.

:FMTSCREEN Syntax

```
>>---:FMTSCREEN --+-panel----+-----+-----------+-----><
                  +-variable-+     +-NEXT------+
```

where:

panel is the repository name of a FMT-SCREEN member

variable is any local, global or command variable that must contain the MP-AID name of the FMT-SCREEN member.

INCLUDE

The :INCLUDE macro includes the contents of the CONTENTS clause of an existing SEXEC member in the current SEXEC member.

The :INCLUDE macro can only be used in SEXEC members.

For example, if you specify:

```
:INCLUDE EM1010
```

the contents of the CONTENTS clause of the SEXEC member EM1010 will be included at the position indicated by the :INCLUDE macro.

INCLUDE Syntax

```
>>---:INCLUDE member-name ----------------------------><
```

where *member-name* is the repository name of an existing SEXEC member.

LPARM

The :LPARM macro sets two variables to the values of the last two parameters received by the executive routine.

The :LPARM macro can be used in SEXEC and LIFE-CYCLE-OBJECT-TYPE members.

The value of the last two parameters received by the executive routine is transferred to &P0 and &P1 or to two specified variables.

For example, if you specify:

```
:LPARM
IF &P0 EQ MDL_SUF1 AND &P1 EQ MDL_SUF2 THEN DO
   instruction_1
   instruction_n
END
```

&P0 is set by :LPARM to the value of the last but one parameter, and &P1 to the value of the last parameter received by the Executive Routine.

For example, if you specify:

```
:LPARM TO MDL_MODE MDL_MEMBER
IF MDL_MEMBER = ' ' THEN -
   instruction_1
```

the local variable MDL_MODE is set by :LPARM to the value of the last but one parameter, and MDL_MEMBER is set to the value of the last parameter received by the Executive Routine.

LPARM Syntax

```
>>──:LPARM ──┬──────────────────────────┬──────────────><
             └─TO variable1 variable2   ┘
```

where:

variable1 is any local, global, or command variable

variable2 is any local, global, or command variable

NAMKO

The :NAMKO macro checks the naming conventions of specific members/relationships and member/relationship types.

The :NAMKO macro can be used in SEXEC and LIFE-CYCLE-OBJECT-TYPE members.

If the naming convention is found to be valid, additional information about the member/relationship or member/relationship type is automatically returned by :NAMKO. This information is placed in command variables having these names:

MDC_NAMING_MTYPE_*name*_*udsname*

where *name* is a letter representing one of these names for the member/relationship type:

- E ENCODE-KEYWORDS
- I INTERROGATE-KEYWORDS
- S STANDARD-LITERAL
- H SHORT-LITERAL

where *udsname* is the name of the UDS-TABLE that contains the member/relationship type.

:NAMKO also returns the following:

- MDC_NAMING_ALIAS, containing the first alias of the member/relationship type
- MDC_NAMING_CONV, an array that contains the naming convention(s) for the member/relationship type
- MDC_NAMING_EXIT, containing the MP-AID name of the local exit, if one has been specified

An example command variable name is MDC_NAMING_MTYPE_I_DU016.

Refer to Chapter 9, "Member Types," on page 215 for details of the MEMBER-TYPE and RELATIONSHIP-TYPE member types.

If the input does not exist, the specified naming convention is invalid: the system variable for return codes (&CCOD) returns with a value greater than zero, and the command variables are set to empty strings.

To check the naming convention of a specific member/relationship type, enter:

```
:NAMKO MT long-name
```

where *long-name* is the value of the LONG-NAME clause for the member/relationship type.

For example, if you specify:

```
:NAMKO MT DB2-TABLE
```

the member type DB2-TABLE is checked. If it exists, the command variable:

- MDC_NAMING_ALIAS is set to the alias of the specified member type, for instance D7
- MDC_NAMING_CONV is set to the naming convention of members of the specified type, for instance TB-___
- MDC_NAMING_EXIT is set to the MPAID-NAME of a local exit, if such an exit has been specified for the member type.

and information about all the names for the member/relationship type, apart from the LONG-NAME, is returned.

For details of checking member/relationship names, refer to Chapter 7, "User Exits," on page 171.

You can specify that only one name for the member/relationship type is tested, by entering:

```
:NAMKO MT name name-type
```

where:

name is the value of one of these names for the member/relationship type:

- ENCODE-KEYWORDS
- INTERROGATE-KEYWORDS
- STANDARD-LITERAL
- SHORT-LITERAL

name-type is a letter indicating which identifier keyword of the member type is to be tested, as follows:

- E ENCODE-KEYWORDS
- I INTERROGATE-KEYWORDS
- S STANDARD-LITERAL
- H SHORT-LITERAL

For example, to test the INTERROGATE-KEYWORDS of the member type PROCESSING-RULE, enter:

```
:NAMKO MT PROCESSING-RULE I
```

To specify the alias of a member/relationship type to be checked, enter:

```
:NAMKO MTK alias
```

where *alias* is the alias of a member/relationship type, as defined in the first ALIAS clause of the MEMBER-TYPE or RELATIONSHIP-TYPE definition.

For example, if you specify:

```
:NAMKO MTK D7
```

the member type alias D7 will be checked. If it exists, the command variable:

- MDC_NAMING_MTYPE is set to the keywords of the corresponding member type, for instance DB2-TABLE
- MDC_NAMING_CONV is set to the naming convention of the corresponding member type, for instance TB-___
- MDC_NAMING_EXIT is set to the MPAID-NAME of a local exit, if such an exit has been specified for the member type.

To check a specific member or relationship, enter:

```
:NAMKO NAME member-name
```

where *member-name* is the name of a particular member or relationship.

For example, if you specify:

```
:NAMKO NAME TB-P01
```

the member name TB-P01 will be checked. If it exists, the command variable:

- MDC_NAMING_ALIAS is set to the alias of the corresponding member type, for instance D7
- MDC_NAMING_MTYPE is set to the keywords of the corresponding member type, for instance DB2-TABLE

To check those members and relationships that have a particular naming convention, enter:

```
:NAMKO NAMK naming-convention
```

where *naming-convention* is the naming convention for the member/relationship type, as defined in the NAMING clause of the MEMBER-TYPE or RELATIONSHIP-TYPE definition.

For example, if you specify:

```
:NAMKO NAMK TB-_0_
```

the string TB-_0_ defines those parts of a naming convention, from which the syntax for a name- related-selection of the LIST command will be generated.

Note:

The string that specifies the naming convention must contain a valid naming prefix (TB-), which indicates the member/relationship type.

After successful generation MDC_NAMING_ALIAS will contain the selection criteria of the LIST command.

To execute the generated LIST command, enter:

```
MPR LIST MDC_NAMING_ALIAS;
```

For example, if you specify:

```
:NAMKO NAMK TB-_0_
MPR LIST MDC_NAMING_ALIAS;
```

all members with the specified naming convention (TB-_0_) are listed.

To check a specific naming convention, enter:

```
:NAMKO NAMKI naming-convention
```

where *naming-convention* is the naming convention for the member/relationship type, as defined in the NAMING clause of the MEMBER-TYPE or RELATIONSHIP-TYPE definition.

For example, if you specify:

```
:NAMKO NAMKI TB-___
```

the naming convention TB-___ will be checked. If it exist, the command variable:

- MDC_NAMING_ALIAS is set to the alias of the corresponding member type, for instance D7
- MDC_NAMING_MTYPE is set to the keywords of the corresponding member type, for instance DB2-TABLE
- MDC_NAMING_EXIT is set to the MPAID-NAME of a local exit, if such an exit has been specified for the member type

NAMKO Syntax

```
>>──:NAMKO ──┬─MT long-name──────────────────┬──────────────────────><
             ├─MT name name-type─────────────┤
             ├─MTK alias─────────────────────┤
             ├─NAME member-name──────────────┤
             ├─NAMK naming-convention────────┤
             └─NAMKI naming-convention───────┘
```

where:

long-name is the value of the LONG-NAME clause for the member/relationship type

name is the value of one of the following names for the member/relationship type: ENCODE-KEYWORDS, INTERROGATE-KEYWORDS,STANDARD-LITERAL, SHORT-LITERAL.

name-type is a letter indicating which identifier keyword of the member type is to be tested, as follows:

- E ENCODE-KEYWORDS
- I INTERROGATE-KEYWORDS
- S STANDARD-LITERAL
- H SHORT-LITERAL

alias is the alias of a member/relationship type, as defined in the first ALIAS clause of the MEMBER-TYPE or RELATIONSHIP-TYPE definition.

member-name is the name of a particular member or relationship.

naming-convention is the naming convention for the member type, as defined in the NAMING clause of the MEMBER-TYPE or RELATIONSHIP-TYPE definition.

NAMKOT

The :NAMKOT macro checks if the naming convention of a specified member is in accordance with the naming convention of its specified type.

The :NAMKOT macro can be used in SEXEC and LIFE-CYCLE-OBJECT-TYPE members.

If the naming convention of a member is in accordance with the naming convention of its member type:

- MDC_NAMING_ALIAS will be set to the member type alias
- MDC_NAMING_MTYPE will be set to the encode keyword of the member type
- MDC_NAMING_CONV will be set to the naming convention of the member type
- MDC_NAMING_EXIT will be set to the MPAID-NAME of a local exit, if such an exit has been specified for the member type

If the naming convention of a specified member is not in accordance with the naming convention of its specified type, :NAMKOT provides several keywords to control the further processing of the member.

For example, if you specify:

```
:NAMKOT FRED ITEM ERROR GOTO OUT
 ...
-OUT
 ...
```

control of the further processing passes to the directives following the OUT label.

For example, if you specify:

```
:NAMKOT FRED ITEM ERROR EXIT
```

the EXIT directive causes an immediate exit from the Executive Routine.

For example, if you specify:

```
:NAMKOT FRED ITEM ERROR CEXEC EC2244
```

CEXEC indicates that control of the further processing will be passed to a Corporate Executive Routine defined in the SEXEC member EC2244.

For example, if you specify:

```
:NAMKOT FRED ITEM ERROR TRANSFER EC3355
```

the TRANSFER directive causes the current Executive Routine and all higher level Executive Routines to be exited from without further processing. The command variables will be RELEASEd, and the Executive Routine defined in SEXEC member EC3355 will be started.

NAMKOT Syntax

```
>>──:NAMKOT name type ERROR ──┬─GOTO label ──────────┬──>>
                              ├─EXIT ────────────────┤
                              ├─CEXEC exec ──────────┤
                              └─TRANSFER exec ───────┘
```

where:

name is the member name.

type is the keyword of a member type, as defined in the ENCODE-KEYWORDS clause of the MEMBER-TYPE.

exec is the name of an SEXEC or EXECUTIVE-ROUTINE member.

OUTE

The :OUTE macro determines the buffer which is to be addressed by an Executive Routine.

The :OUTE macro can be used in SEXEC and LIFE-CYCLE-OBJECT-TYPE members.

The macro provides the same functionality as the SET OUTPUT-EDIT ON/OFF command but instead of the complete syntax you just enter:

```
:OUTE ON
```

or

```
:OUTE OFF
```

Refer to *ASG-ControlManager User's Guide* for details of the SET OUTPUT-EDIT command.

OUTE Syntax

```
>>──:OUTE ──┬─ON ─┬──────────────────────────────────────><
            └─OFF ┘
```

RETAIN

The :RETAIN macro retains an Executive Routine in virtual storage.

The :RETAIN macro can be used in SEXEC and LIFE-CYCLE-OBJECT-TYPE members.

If the :RETAIN macro is specified the executive routine is retained until you log off from ASG software.

To improve performance, only those Executive Routines should be retained that are frequently used.

Note:

Use the global variables MDG_RETAIN1 to MDG_RETAIN9 to activate or inactivate a certain class of :RETAIN macro. For details of global variables, refer to Chapter 6, "Customizing the Environment," on page 103.

You can either retain an Executive Routine in general or with a specified class that indicates the priority with which the routine is to be retained.

For example, if you specify:

```
:RETAIN YES
```

the Executive Routine will be retained in general.

For example, if you specify:

```
:RETAIN CLASS=1
```

the Executive Routine will be retained with the highest priority in virtual storage.

Note:
The :RETAIN macro has to be specified at the beginning of an Executive Routine.

RETAIN Syntax

```
>>──:RETAIN ┬─YES─────┬──────────────────────────────────><
            └─CLASS=n─┘
```

where *n* is one of these:

- For MMR/TSS routines with high usage
- For MMR/TSS routines with medium usage
- For naming convention tables
- For member type cluster menus
- For assisted update
- For MMR/LCS routines with high usage
- For MMR/LCS routines with medium usage
- Other MMR/LCS components
- Other MMR/TSS components

VCHNG

The :VCHNG macro replaces all or specified occurrences of one character string by another in a selection of a variable's elements.

The :VCHNG macro can be used in SEXEC and LIFE-CYCLE-OBJECT-TYPE members.

The syntax of the macro is similar to the primary command CHANGE. In contrast to the primary command, that operates on a buffer, :VCHNG operates on a variable.

For example, if you specify:

```
:VCHNG MDC_AUPD_BUF /MDL_OLD/MDL_NEW/ MDL_START MDL_LINO *
```

the macro operates on the command variable MDC_AUPD_BUF. This variable contains an array. In the array an old string, set to MDL_OLD, is to be replaced by a new string, set to MDL_NEW. MDL_START is set to an element number, that indicates the start for the replacement in the array. MDL_LINO is set to the total number of elements that are to be checked for changes. The asterisk (*) indicates that all occurrences of the old string are to be replaced by the new string for the specified number of elements.

VCHNG Syntax

```
:VCHNG var1 /old/new/ var2 ──┬─ mm ─┬─┬─ nn ─┬──
                             └─ * ──┘ └─ * ──┘
```

where:

var1 is any local, global, or command variable.

old is a character string to be changed or deleted.

new is a character string to be inserted. If the old character string is to be deleted, the variable set to the new string must be empty.

var2 is any local, global, or command variable that is set to the element number of an array. This number specifies the position, where the replacement/deletion starts.

mm is the total number of elements that are to be checked for changes or deletions. mm can be a variable or a constant.

nn is the number of occurrences of the string which are to be changed or deleted on each element. *nn* can be a variable or a constant.

* denotes all.

VSEARCH

The :VSEARCH macro searches through a variable's elements for a given string.

The :VSEARCH macro can be used in SEXEC and LIFE-CYCLE-OBJECT-TYPE members.

For example, if you specify:

```
:VSEARCH MDG_OUT /MDL_STR/ MDL_POS
```

the macro operates on the global variable MDG_OUT. This variable contains an array. The elements of the array are searched for the string, MDL_STR is set to. MDL_POS is set to an element number that indicates the start for the search in the array. If the string can be located, MDL_POS will be set automatically to the number of the element that contains the string. Otherwise the value of MDL_POS will remain its start value.

VSEARCH Syntax

```
>>──:VSEARCH variable1 /string/ variable2──────────────────────><
```

where:

variable1 is any local, global, or command variable

string is a character string to be located

variable2 is any local, global, or command variable that is set to the element number of an array. This number specifies the position, where the search for the string starts. If the string can be located variable2 will be set to the number of the element that contains the string. Otherwise its value will not change.

XFILE

The :XFILE macro files the contents of one or more variables in the source record of a member.

The :XFILE macro can be used in SEXEC and LIFE-CYCLE-OBJECT-TYPE members.

Whether the source record that receives the contents of the variable(s) is an existing record or a new record, and whether the source record is to be encoded, depends on the primary command (ALTER, MODIFY, REPLACE, or INSERT) specified in the :XFILE macro.

For details of the commands, refer to *ASG-Manager Products Dictionary/Repository User's Guide*.

In the first step, the :XFILE macro deletes the complete source record of the specified member, if that member already exists. In the second step, :XFILE writes the contents of the specified variable(s) to the empty source record of the specified member.

If the member is to be encoded, it must receive a complete source record in the original Manager Products syntax from the specified variable(s).

For example, if you specify:

```
:XFILE MODIFY FRED MDG_SUBSTI
```

the global variable MDG_SUBSTI must contain the complete source record that is to be written to the member FRED. For instance, if FRED is an ITEM, the source must include all clauses and their corresponding entries, which are mandatory for this member type. The member will then be automatically encoded, because MODIFY has been specified.

If MDG_SUBSTI contains an incomplete source record, the encode will fail and the process will be aborted. ASG recommends that you check the return code of the primary command, because the :XFILE macro outputs neither a message nor a report.

For example, if you specify:

```
:XFILE MODIFY FRED MDG_SUBSTI
IF &ccod > 4 THEN -
   MPR ENCODE FRED;
ELSE DO
   :BROWSE
   SAY #SOURCE SUCCESSFULLY TRANSFERRED AND ENCODED#
END
```

a report will be output, as a result of the ENCODE command that indicates errors in the source record, if the encode fails (&ccod > 4). Otherwise a message displays.

Note:

The hash characters (#) are used as literals in the example.

You can also specify several variables, each containing a certain part of a source, that will be written to the source record of the specified member.

For example, if you specify:

```
:XFILE ALTER FRED MDL_SUB1 MDL_SUB2 MDL_SUB3
```

the contents of the local variables (MDL_SUB1, MDL_SUB2, MDL_SUB3) will be written to FRED's source record in the same sequence as stated in the macro.

You can specify as many variables as you like in an :XFILE macro, for instance one variable for each clause and its entries.

If the source record of the specified member is composed of the contents of several variables, you must ensure that the resulting source record is a complete one. Otherwise it can not be encoded.

In the last example the encode is not done automatically, because ALTER has been specified.

XFILE Syntax

```
                                                <<<<<<<<<
>>──:XFILE ──┬─ALTER  ──┬─member-name variable ──────────><
             ├─MODIFY ──┤
             ├─REPLACE ─┤
             └─INSERT ──┘
```

where:

member-name is a repository member name.

variable is any local, global or command variable.

9 Member Types

This chapter includes these sections:

ATTRIBUTE-GROUP 218
Specifying the ATTRIBUTE-TYPE members Contained in the Group. 219
ATTRIBUTE-GROUP Syntax 220

ATTRIBUTE-TYPE 220
Defining the Name of a Clause or Identified Keyword 221
Defining the Name of an Unidentified Keyword 222
Defining the Type of Value Permitted 222
Defining Specific Permitted Values 225
Defining the Permitted Number of Values 226
Defining Minimum and Maximum Permitted Values 227
Defining Installation Independent Date and Time Values 227
Defining the Length of a Value. 229
Defining the Number of Lines of Text that can be Entered in a Clause 229
Indexing User-defined Attributes by Presence or Value. 229
Renaming UDR and UDRS Clauses and Displaying Clauses with Identifiers Containing More than One Keyword. 231
Defining a Line of Help in an Assisted Update Buffer. 233
Defining an Assisted Update Buffer Input Prompt 233
Defining a Complex Assisted Update Buffer Input Prompt 235
Defining How Clauses and Keywords are Formatted by Assisted Update. 236
Taking a User Exit Defining how Clauses and Keywords are Formatted by Assisted Update 239
Displaying Repeating Clauses and Keywords in Assisted Update 239
Defining When Clauses and Keywords are Displayed in an Assisted Update Buffer . 240
Documenting Help for a Clause or Keyword 241
ATTRIBUTE-TYPE Syntax 241

FMT-SCREEN 247
Defining the Help for the Panel 248
Defining an MP-AID Name for a FMT-SCREEN Member 250
Defining the Panel Type 250
Defining a Point of Return for the Control Program. 250
Defining the Appearance of the Panel When Returned to From Another Panel 251
Defining Field Control Characters 251
Defining Input and Output Fields in the FMT-SCREEN Member. 254
Specifying a Relationship to ITEM Members Defining Output Fields 255

Specifying a Relationship to ITEM members Defining Input Fields 255
Defining the Processing of the Panel 255
Defining a Command Area 261
Defining the Position of the Function Key Area. 262
Defining the Allowed User Actions for the Panel 262
Defining the Position of the Message Area. 263
Defining a One-line Header 263
Defining the Layout of the Panel 263
FMT-SCREEN Syntax 264

HDS-TABLE 268
Specifying the Member Types for Generation 269
Specifying the Relationship Types for Generation 269
Specifying a Name for the Generated HDS Table 269
Specifying the RIM for Generation. 269
Specifying a Name for the Generated Translation Executive Routine 270
Including User-Defined EA Relationships in the Generation. 270
Example. 270
HDS-TABLE Syntax 270

HIERARCHY 271
Naming the MP-AID Members Generated from the RIM 271
Specifying the Entity Member Types Contained in the RIM 272
Assigning Values to Entity Member Types 273
Specifying the Relationship Member Types Included in the RIM 273
Defining Mutually Exclusive Relationship Member Types 274
Assigning Values to Relationship Member Types 275
Defining Collective Member Types 276
Specifying the User-defined Attributes Common to all Member Types 277
Assigning Parameter, Line, and Format Line Numbers to User-defined Attributes. . . 278
Specifying the UDR and UDRS Clauses to be Included in the RIM 279
HIERARCHY Syntax 279

INFOBANK-PANEL 282

ITEM 282
Defining a Title 283
Defining Lower, Upper or Mixed Case Mode 283
Defining Valid Input Values 283
Defining the Form of the Data 284
Defining Help 285
ITEM Syntax. 285

MEMBER-TYPE 286
Defining a Base or User-defined Member Type 286
Defining the Keywords With Which the Member Type is Encoded 287
Defining Keywords With Which the Member Type can be Interrogated. 288
Defining Keywords That Can Be Specified in a REPORT DOWN-TO Command. . . 289
Tailoring GLOSSARY, REPORT, WHAT, and WHICH Output 289
Tailoring LIST Output 290

Tailoring SHOW UDS Output 291
Tailoring GLOSSARY and LIST Headings and Totals Output 291
Specifying Generic User-defined Attributes 292
Specifying Non-Generic User-defined Attributes 293
Defining a Member Type Level Number 294
Disallowing Relationships Between Members of the Same Member Type 294
Allowing and Disallowing Relationships Via Specified Clauses 295
Automatically Defining EA Relationships 296
Preventing a Member Type Being Displayed in the Panel Interface/Displaying IMS Collective Member Types 297
Defining Naming Conventions for Entity Members 298
Defining Complex Naming Conventions 301
Specifying the Clauses and Keywords Displayed During Assisted Update 302
Defining an Alias Identifier 302
Documenting Help for a Member Type 303
MEMBER-TYPE Syntax 303

MEMBER-TYPE-GROUP 306
Specifying the Entity Member Types Contained in the Group 306
Defining a Member Type Cluster Menu Option 307
Specifying the Member Types Selected from the Cluster Menu 308
MEMBER-TYPE-GROUP Syntax 309

RELATIONSHIP-CLASS 310
RELATIONSHIP-CLASS Syntax 311

RELATIONSHIP-GROUP 311
Defining a Group of Relationship Member Types 311
Defining Mutually Exclusive Relationship Member Types 312
RELATIONSHIP-GROUP Syntax 313

RELATIONSHIP-TYPE 314
Naming the Relationship Member Type 315
Defining the Relationship Type Class 316
Tailoring GLOSSARY, REPORT, WHAT, and WHICH Output 317
Tailoring LIST Output 317
Tailoring LIST and GLOSSARY Headings Output 318
Defining the Source and Target Member Types 318
Disallowing Unencoded Source and Target Members 319
Defining a Permitted Number of Relationships 319
Making Relationships via the Relationship Member Type Mandatory 320
Controlling the Removal of Members Participating in a Relationship 320
Allowing and Disallowing Duplicate Relationships 321
Allowing a Member to be Both the Source and Target of a Relationship 322
Documenting the Order of Retrieval of Source and Target Members 322
Specifying the User-defined Attributes that can be Included in the Member Type . . . 323
Allowing and Disallowing Relationships Via Specified Clauses 324
Automatically Defining EA Relationships 325
Defining Naming Conventions for Relationship Members 326
Taking a User Exit Defining Complex Naming Conventions 330

Preventing a Member Type Being Displayed in the Panel Interface 330
Specifying the Clauses and Keywords Displayed During Assisted Update 331
Defining an Alias Identifier 332
Documenting Help for a Member Type 332
RELATIONSHIP-TYPE Syntax. 332

SEXEC 335

ATTRIBUTE-GROUP

The ATTRIBUTE-GROUP member type defines a group of ATTRIBUTE-TYPE members. Refer to "ATTRIBUTE-GROUP Syntax" on page 220 for the syntax. Each of the ATTRIBUTE-TYPE members in the group defines:

- A user-defined attribute (a user-defined clause or keyword)

and/or:

- The format in which a user-defined or ASG-supplied clause or keyword displays in the panel interface

By defining an ATTRIBUTE-GROUP member you can reference ATTRIBUTE-TYPE members collectively and so simplify your RIM definition.

For example you can specify:

- The clauses and keywords to be displayed during an assisted update of a member type by defining an ATTRIBUTE-GROUP member and specifying it in the SEE clause of the relevant MEMBER-TYPE member
- The user-defined attributes common to all member types in the RIM by defining an ATTRIBUTE-GROUP member and specifying it in the COMMON-ATTRIBUTES clause of the HIERARCHY member

You can define mutually exclusive sets of user-defined attributes. Only one of the alternative attributes can be specified in the definition of a member.

You can also define whether a user-defined attribute must be specified in the definition of a member.

You cannot specify that ASG-supplied clauses and keywords are mutually exclusive or optional as this is predefined and cannot be tailored.

Specifying the ATTRIBUTE-TYPE members Contained in the Group

To define a group of ATTRIBUTE-TYPE members, specify:

`CONTAINS` *`attribute-list`*

where *`attribute-list`* is:

- The names of one or more ATTRIBUTE-TYPE or ATTRIBUTE-GROUP members. If an ATTRIBUTE-GROUP member is specified then all the ATTRIBUTE-TYPEs it directly and indirectly contains are included in the group. Each member name must be separated by a comma if entered via the command interface.

and/or:

- One or more sets of mutually exclusive ATTRIBUTE-TYPE member names. Each set must contain two or more ATTRIBUTE-TYPE member names each separated by an ELSE keyword. The same ATTRIBUTE-TYPE member name can be specified in more than one set. Each set must be separated by a comma if entered via the command interface.

Each individual ATTRIBUTE-TYPE or ATTRIBUTE-GROUP member name or each set of alternative ATTRIBUTE-TYPE members can be followed with:

OPTIONAL NO to specify that:

- A user-defined attribute or all the attributes contained in the named group
- One of the mutually exclusive attributes within a set must be present when a member is encoded.

OPTIONAL YES to specify that the attribute(s) need not be present.

OPTIONAL WARN to specify that the attribute(s) need not be present but, if not a warning message displays.

The default is OPTIONAL YES.

The CONTAINS clause is optional.

ATTRIBUTE-GROUP Syntax

```
>------ATTRIBUTE-GROUP ------------------------------------------------>
>-------------------------------------------------------------------------->
  |                <<<<<<<<<<<<<<<<<<<<<<<<<<< , <<<<<<<<<<<<<<<<<<<<<<<<<<<
  |-CONTAINS -|-a-type -------------------------------------------------|
  |           |         |<<<<<<<<<<<<<<<<<<
```

The type of values permitted in ASG-supplied clauses and keywords is predefined and cannot be tailored via an ATTRIBUTE-TYPE member. For ASG-supplied clauses and keywords the ATTRIBUTE-TYPE member definition need only contain those clauses that identify the clause or keyword (IDENTIFIED-BY or NAMED), its attribute type (for example, CHARACTER-STRING) and define how it is to be processed by the panel interface (for example, EDIT-CODE-1 and SKELETON-HELP). For ASG-supplied unidentified keywords you also have to specify a VALUES clause: enter one or more of the default values in it.

Refer "ATTRIBUTE-TYPE Syntax" on page 241 for the syntax of the ATTRIBUTE-TYPE member type.

Defining the Name of a Clause or Identified Keyword

To define the name of a clause or identified keyword, specify:

```
IDENTIFIED-BY identifier
```

where *identifier*:

- Can contain up to 32 characters from the standard character set for member names but must not begin with an underscore
- Must not be the same as the identifier of any other clause or keyword also available in the definition of the member type in which the defined clause or keyword is permitted.

Both the identifier and the values of a clause or keyword must be specified in the definition of a member.

The identifier is displayed above any input prompts in an assisted update buffer.

If the identifier is that of an ASG-supplied clause or keyword then the supplied defaults will apply.

If you are renaming UDR and UDRS clauses then:

- The ASG-supplied clause name (UDR1 to UDR9 and UDRS) must be specified in the IDENTIFIED-BY clause
- The user-defined clause name must be specified in the LONG-NAME clause.

The IDENTIFIED-BY clause is mandatory for all ATTRIBUTE-TYPE members except those having an unidentified KEYWORD attribute type for which a NAMED clause must be specified.

Refer to *ASG-ControlManager User's Guide* for details of the character set for names.

Defining the Name of an Unidentified Keyword

To define the name of an unidentified keyword, specify:

```
NAMED identifier
```

where *identifier*:

- Can contain up to 32 characters from the standard character set for member names but must not begin with an underscore
- Must not be the same as the identifier of any other clause or keyword also available in the definition of the member type in which the defined keyword is permitted

If the identifier is that of an ASG-supplied clause or keyword then the supplied defaults will apply.

Only the values (defined in the VALUES clause) of an unidentified keyword must be specified in the definition of a member. No identifier is required.

The identifier is displayed above any input prompts in an assisted update buffer. You must specify a PROMPT-CODE O or P clause so that the when the keyword is updated via an assisted update buffer its identifier is deleted when a member is filed in the repository.

For example, if you had defined an ATTRIBUTE-TYPE member containing the following clauses:

```
KEYWORD
NAMED SECURITY-STATUS
VALUES CONFIDENTIAL
PROMPT-CODE O
```

then the definition of a member containing the keyword would include its value CONFIDENTIAL and exclude its identifier SECURITY-STATUS.

The identifier is the name with which an unidentified keyword can be interrogated, for example in WHICH MEMBERS HAVE commands.

The NAME clause is mandatory for ATTRIBUTE-TYPE members with an unidentified KEYWORD attribute type.

Defining the Type of Value Permitted

It is mandatory to specify one of the keywords. The attribute type of ASG-supplied clauses and keywords are predefined and cannot be tailored. You should specify a keyword that matches this default when creating an ATTRIBUTE-TYPE member to define how an ASG-supplied clause or keyword is displayed in the panel interface. Use the SHOW command to find out the default attribute type.

The type of values that the above keywords permit is described below. You cannot encode a member if its definition contains a clause or keyword with a value that is not of the permitted type.
To define the type of data permitted as the value of a clause or keyword, specify one of these attribute-types:

Attribute-type	Definition
CHARACTER-STRING	A string of up to 256 characters. If the value consists of two or more delimited strings, the strings are automatically concatenated when a member containing the clause is encoded. The value must be delimited if entered via the command interface or specified in the VALUES, MINIMUM-VALUE or MAXIMUM-VALUE clauses.

DATE	A date in the format defined in the DCUST installation macro. By default this is:
	day month year
	where:
	day is one/two numeric characters in the range 1 or 01 to 31.
	month is either:
	• One or two numeric characters in the range 1 or 01 to 12
	• One of the following character strings which may be unambiguously truncated:
	`JAN FEB MAR APR MAY JUN`
	`JUL AUG SEP OCT NOV DEC`
	year is two or four numeric characters in the range 00 or 1000 to 2999, specifying the last two or all four characters of the year.
	By default each element of the date is separated by a space. If the separator is a space or (if you have tailored DCUST) a comma, semi-colon or right round bracket, the date must be delimited if entered via the command interface or specified in the VALUES, MINIMUM-VALUE or MAXIMUM-VALUE clauses.

DECIMAL-NUMBER	Up to 18 digits, optionally preceded by a sign and/or containing a decimal point. The decimal point must not be the final character.

Attribute-type	Definition
FREE-FORM-TEXT	From 1 to 32767 lines of text starting on the line following the keyword. Each line may contain a maximum of 248 characters. Any character can be specified, but if a terminator is detected in the first character position of a line, it is assumed that the end of the member definition has been reached. Only one clause with a FREE-FORM-TEXT value can appear in any member definition and, if specified, must come at the end of the definition.
INTEGER	Up to 18 digits, optionally preceded by a sign.
KEYWORD	A keyword containing up to 32 characters from the standard character set for repository member names but which must not begin with an underscore. If the ATTRIBUTE-TYPE member also contains a NAMED clause then the value will define an unidentified keyword (a keyword not preceded by an identifying keyword when entered in a member definition). Alternatively if the ATTRIBUTE-TYPE member contains an IDENTIFIED-BY clause then the value will define an identified keyword that must be preceded by the keyword defined in that clause.
NAME	A name that obeys the rules for repository member names. Values: containing only numeric characters or any character from the extended character set beginning with an underscore or hyphen must be delimited if entered via the command interface or specified in the VALUES, MINIMUM-VALUE or MAXIMUM-VALUE clauses.
TEXT	Up to 32767 character strings. Each string can contain up to 246 characters. The value must be delimited if entered via the command interface. If a member containing a clause with a text value is processed by a REPORT or GLOSSARY command, then the strings are not concatenated but are aligned one beneath the other on the opening delimiters, even when several strings appear on a single input line. This allows you to include, for example, tables and lists in a member's definition.
TIME	A time value in the format defined in the DCUST installation macro. By default this is:

Attribute-type	Definition
	`hour minute second` where: `hour` may be one or two numeric characters in the range 0 or 00 to 24. `minute` may be one or two numeric characters in the range 0 or 00 to 59. `second` may be one or two numeric characters in the range 0 or 00 to 59. The inclusion of seconds is optional. By default each element of the time is separated by a space. If the separator is a space or (if you have tailored DCUST) a comma, semi-colon or right round bracket, the time must be delimited if entered via the command interface or specified in the VALUES, MINIMUM-VALUE or MAXIMUM-VALUE clauses.

Defining Specific Permitted Values

To define the specific values that are permitted for a user-defined attribute, specify:

```
VALUES value-list
```

where `value-list` can be one or more of the values permitted with any of these attribute types:

- CHARACTER-STRING
- DATE
- INTEGER
- KEYWORD
- NAME
- TIME

Refer to "Defining the Type of Value Permitted" on page 222 for details of the values permitted for the above attribute types.

A maximum of 255 KEYWORD values can be specified.

Each value in the value-list must be separated by a comma if entered via the command interface.

The listed values must not conflict with the restrictions specified in the MULTIPLE-VALUES, MAXIMUM-LENGTH, MINIMUM-LENGTH, MAXIMUM-LINES, and MINIMUM-LINES clauses.

A member containing an attribute whose value is not one of those permitted will not encode successfully. For example, if you defined the following integer value:

```
VALUES 12345
```

then a member containing the attribute would only encode if it had a value of 12345.

A member can contain more than one value if you have also specified a MULTIPLE-VALUES clause in the ATTRIBUTE-TYPE member definition.

The VALUES clause is mandatory for KEYWORD attributes and optional for attributes with any other attribute type. You can alternatively specify a NORMALIZED-VALUE clause for user-defined DATE and TIME attributes.

Defining the Permitted Number of Values

To define that a user-defined attribute can contain more than one value, specify:

```
MULTIPLE-VALUES
```

To define the minimum and maximum number of values permitted for a user-defined attribute, specify:

```
MULTIPLE-VALUES MINIMUM-NUMBER integer
                MAXIMUM-NUMBER integer
```

where *integer* is an integer in the range 1 to 32767.

A member containing an attribute with too few and/or too many values will not encode successfully.

For example, if you defined an ATTRIBUTE-TYPE member containing the following clause:

```
MULTIPLE-VALUES MINIMUM-NUMBER 2
```

then a member containing the attribute would only encode if it had two values.

In order to update an attribute with multiple values via an assisted update buffer you must also define EDIT-CODE-1 and EDIT-CODE-2 clauses with values of 3, 5, or 8 in the definition of the ATTRIBUTE-TYPE member.

Each value specified in a members definition must be separated by a comma if entered via the command interface.

The MULTIPLE-VALUES clause is optional. A user defined attribute can only contain a single value if the MULTIPLE-VALUES clause is not specified. The MULTIPLE-VALUES clause cannot be specified for TEXT or FREE-FORM-TEXT attributes.

Defining Minimum and Maximum Permitted Values

To define the minimum and maximum permitted value for a user-defined attribute, specify:

```
MINIMUM-VALUE value
MAXIMUM-VALUE value
```

where *value* can be any of the values permitted with the following attribute types:

- TIME
- DATE
- INTEGER
- NAME
- CHARACTER-STRING

Refer to "Defining the Type of Value Permitted" on page 222 for details of the permitted values.

For example, if you defined minimum and maximum character string values as follows:

```
MINIMUM-VALUE A
MAXIMUM-VALUE C
```

then a member containing the defined attribute would only encode if it had a value of A, B, or C.

The MINIMUM-VALUE and MAXIMUM-VALUE clauses are optional.

You can alternatively specify NORMALIZED-MINIMUM-VALUE and NORMALIZED-MAXIMUM-VALUE clauses for DATE and TIME attributes

Defining Installation Independent Date and Time Values

To define (in a format independent of that defined in the DCUST installation macro) the values permitted for user-defined DATE and TIME attributes, specify:

```
NORMALIZED-VALUES value
```

where *value* is:

- *hhmmss* for TIME attributes

where:

hhmmss is six numeric characters in the range 000000 to 240000.

hh is the hour

mm is the number of minutes, *ss* is the number of seconds.

- *yyyyddd* for DATE attributes

  where:

  yyyy is four numeric characters specifying the year

  ddd is three numeric characters (in the range 001 to 365) specifying the number of the day within the year.

More than one value can be specified. Each value must be separated by a comma if entered via the command interface.

For example, if you defined the following date value:

```
NORMALIZED-VALUE 1992005
```

then a member containing the defined attribute would only encode if the attribute had a value for the fifth of January 1992 in the format specified in the DCUST installation macro.

To define installation independent minimum and maximum permitted values, specify:

```
NORMALIZED-MINIMUM-VALUE value
NORMALIZED-MAXIMUM-VALUE value
```

The NORMALIZED-VALUE, NORMALIZED-MINIMUM-VALUE, and NORMALIZED-MAXIMUM-VALUE clauses are optional. They are useful if you:

- Do not know the formats specified in the DCUST installation macro
- Want to define attributes for use across several repositories, each of which could have different formats specified for dates and times in the DCUST macro

You can alternatively specify date and time values using the VALUES, MINIMUM-VALUE, and MAXIMUM-VALUE clauses.

Defining the Length of a Value

To define the minimum and maximum length of a user-defined attribute, specify:

```
MINIMUM-LENGTH n
MAXIMUM-LENGTH n
```

where *n* is an integer in the range:

- 1 to 18 for DECIMAL-NUMBER and INTEGER attribute types
- 1 to 246 for TEXT attribute types
- 1 to 248 for FREE-FORM-TEXT attribute types
- 1 to 32 for NAME attributes types
- 1 to 256 for CHARACTER-STRING attribute types

A member definition cannot be encoded if it contains an attribute with a value that is not of the permitted length.

The length of DATE and TIME values are defined in the DCUST installation macro. KEYWORD values can have a length of from 1 to 32 characters.

The MINIMUM-LENGTH and MAXIMUM-LENGTH clauses are optional.

Defining the Number of Lines of Text that can be Entered in a Clause

To define the number of lines that can appear in a user-defined TEXT or FREE-FORM-TEXT attribute, specify:

```
MINIMUM-LINES n
MAXIMUM-LINES n
```

where *n* is an integer not greater than 32767.

The MINIMUM-LINES and MAXIMUM-LINES clauses are optional.

Indexing User-defined Attributes by Presence or Value

To index a user-defined attribute by presence or value, specify:

```
INDEXED-BY PRESENCE
```

or

```
INDEXED-BY VALUE
```

TEXT and FREE-FORM-TEXT attributes cannot be indexed by value. Attributes with any other type of value can be indexed by both presence and value.

CHARACTER-STRING attributes are indexed by presence alone if the character string plus the keyword is more than 78 characters.

DATE and TIME attributes are indexed by the normalized value irrespective of the format in which the value is entered.

DECIMAL-NUMBER attributes are indexed with leading and trailing zeros to 18 digits before the decimal point and 18 places after the decimal point, and a sign.

For example, 1.25 is indexed as:

```
+000000000000000001.250000000000000000
```

This is to ensure that interrogations such as:

```
WHICH MEMBERS HAVE attribute EQUALS 1.25;
```

give the correct numeric value results (so, for example, 1.25, 001.25, and 1.2500 are not treated as different numbers).

INTEGER attributes are indexed with leading zeros to 18 digits and a sign. For example, 125 is indexed as:

```
+000000000000000125
```

This is to ensure that interrogations such as:

```
WHICH MEMBERS HAVE attribute EQUALS 125;
```

give the correct numeric value results (so that, for example, 125, 00125, and +125 are not treated as different numbers).

When a member containing an indexed attribute is encoded, the presence of the attribute is recorded in the repository index dataset. This extends the ways in which the attribute can be processed as follows:

- It appears in the output from the LIST INDEX-NAMES and LIST ATTRIBUTES commands (subject to any other selection criteria in the command)
- It can be selected by the PERFORM and WHICH commands
- It can be identified by the WHAT IS command
- It can be processed by any panel interface functions that internally execute the above commands

In addition, the index dataset is designed to achieve the fastest possible retrieval of names recorded in the dataset. The value of any attribute, whether indexed or not, can be output by the GLOSSARY command or interrogated by the WHICH command. However indexed attributes can be processed faster than those that are not.

If the attribute is indexed by presence alone, a single entry is created on the index dataset containing a pointer to each member with that attribute.

If the attribute is indexed by value, an entry is created containing pointers to the members with that attribute. One or more other entries are created, one for each different value of the attribute, containing pointers to the members containing the attribute with that value.

For example, assume that you have defined an attribute named COLOR which is indexed by value. The attribute may have the values BLUE, RED, or GREEN. When the first member containing the attribute:

```
COLOR RED
```

is encoded, an entry for the attribute COLOR is created on the index dataset and another for the attribute value COLOR=RED, each containing a pointer to that member. If a member containing the attribute:

```
COLOR GREEN
```

is then encoded, the entry for the attribute COLOR is updated to contain a pointer to that member and a new entry is created for the attribute value COLOR=GREEN. If a further member containing the attribute:

```
COLOR RED
```

is encoded, the entries for the attribute COLOR and for the attribute value COLOR=RED are both updated to contain pointers to that member.

The INDEXED-BY clause is optional.

Renaming UDR and UDRS Clauses and Displaying Clauses with Identifiers Containing More than One Keyword

To rename UDR and URDS clauses, specify:

`LONG-NAME` *`udr/udrs-clause`*

where *`udr/udrs-clause`*:

- Is a renamed UDR or UDRS clause name
- Must be delimited if entered via the command interface

The ASG-supplied clause name (UDR1 to UDR9 and UDRS) must be specified in the IDENTIFIED-BY clause.

The renamed UDR clause name displays above the input prompt in an assisted update buffer.

The renamed UDRS clause name is not automatically displayed in an assisted update buffer. To display a renamed UDRS clause you must specify it in the SKELETON-TEXT clause of each of the ATTRIBUTE-TYPE members defining the UDR clauses.

For example, if you had renamed the UDR1 clause to UPDATED-BY and the UDRS clause to UPDATE-PROGRAM then you could display the renamed clauses in an assisted update buffer by defining two ATTRIBUTE-TYPE members containing the following clauses:

```
IDENTIFIED-BY UDR1
LONG-NAME UPDATED-BY
SKELETON-TEXT &P1
              UPDATE-PROGRAM &P1

IDENTIFIED-BY UDRS
LONG-NAME UPDATE-PROGRAM
```

To display a clause identifier made up of multiple keywords, specify:

```
LONG-NAME keyword-list
```

where *keyword-list*:

- Is two or more keywords, each separated by a space
- Is displayed above the input prompt in an assisted update buffer
- Must be delimited if entered via the command interface

Particular ASG-supplied clauses have identifiers made up of multiple keywords. Only one keyword can be specified in an IDENTIFIED-BY clause. In order to update these clauses via an assisted update buffer, you must therefore specify them in a LONG-NAME clause.

For example, to display the IDENTIFIER IS clause of the ENTITY member type, specify:

```
LONG-NAME IDENTIFIER IS
```

The IDENTIFIED-BY clause is mandatory and ASG recommends that you specify in it the first keyword of the keyword-list.

The LONG-NAME clause is mandatory for ASG-supplied clauses having multiple identifying keywords. User-defined attributes can only have a single identifying keyword.

The LONG-NAME clause is optional.

If it is not specified:

- The ASG-supplied UDR and UDRS clause names will apply.
- Those ASG-supplied clauses having multiple identifying keyword cannot be displayed.

Defining a Line of Help in an Assisted Update Buffer

To define a line of help for a clause or keyword in an assisted update buffer, specify:

```
SKELETON-HELP help
```

where *help*:

- Is a string of from 1 to 61 characters
- Must be delimited if entered via the command interface

The help displays in an assisted update buffer alongside the identifier defined for the clause or keyword in the IDENTIFIED-BY, NAMED, or LONG-NAME clause.

The SKELETON-HELP clause is optional.

If you do not define a SKELETON-HELP clause then a default help string displays describing the type of value that can be entered. This help is determined by the clause or keyword's attribute type.

Defining an Assisted Update Buffer Input Prompt

To define an assisted update buffer input prompt for a clause or keyword, specify:

```
SKELETON-CODE code
```

where *code* is an integer in the range 1 to 8.

The codes generate these prompts in an assisted update buffer:

Code	Prompt	Description
1		No prompt displays.
2	??.	Standard prompt. Set in variable DG_SKSTR2
3	??.??	Time prompt. Set in variable MDG_SKSTR3

Code	Prompt	Description
4	??.??.??	Date prompt. Set in variable MDG_SKSTR4
5	?. ??.	Alias prompt. Set in variable MDG_SKSTR5
6	?xxxx	Mandatory input prompt. Set in variable MDG_SKSTR6
7	L??.	Keyword confirmation prompt. Set in variable MDG_SKSTR7
8		No prompt displays. The clause is automatically maintained by the AUTO-REF-STRING clause of the MEMBER-TYPE containing the ATTRIBUTE-TYPE member defining the clause.

You can tailor the default prompts by changing the values set in SEXEC EC1060 for the above variables. You must ensure that variables MDG_SKSTR*n* (where *n* is a code in the range 2 to 7) reflect any changes you have made to the line erase character using variable MDG_DELSTR.

The prompts guide you in the type of value that can be entered for the clause or keyword.

For example, the alias prompt:

```
?. ??.
```

prompts you to enter two separate values such as:

```
SQL DEPARTMENT
```

A clause or keyword with a prompt containing the line erase characters ??. is not filed in the definition of the updated member. The mandatory input prompt (?xxxx) does not contain the string ??. and is therefore not stripped out when the updated member is filed. ?xxxx must be overkeyed by an acceptable value or otherwise the updated member will fail to encode. The mandatory input prompt must not be specified for clauses that can contain free form text as ?xxxx would in that circumstance be an encodable value. You can also support mandatory clauses and keywords in an assisted update buffer by specifying a PROMPT-CODE M clause.

The SKELETON-CODE clause is optional.

If you do not specify a SKELETON-CODE clause then:

- A date prompt displays for clauses with a DATE attributes type
- A time prompt displays for clauses with a TIME attribute type
- A standard prompt displays for clauses and keywords with any other attribute types

You can also define more complex prompts by specifying SKELETON-CODE prompts within a SKELETON-TEXT clause.

Defining a Complex Assisted Update Buffer Input Prompt

To define a complex assisted update buffer input prompt for a clause or keyword, specify:

```
SKELETON-TEXT prompt
```

where *prompt* is one or more strings of text:

- Displayed below the clause or keyword identifier in an assisted update buffer
- Delimited if entered via the command interface

You can incorporate SKELETON-CODE prompts by specifying parameters at the position within the SKELETON-TEXT prompt at which you want the SKELETON-CODE prompt to appear. For example, if you specified:

```
SKELETON-TEXT SQL       &P2
              COBOL     &P2
              ASSEMBLER &P2
```

then the following prompts supporting aliases would be displayed in the assisted update buffer:

```
SQL       ??.
COBOL     ??.
ASSEMBLER ??.
```

These are the different parameters:

Parameter	Prompt	Description
&P1	??.	Set in MDG_DELSTR.
&P2	??.	Set in MDG_SKSTR2. Equivalent SKELETON-CODE 2.
&P3	??.??	Set in MDG_SKSTR3 Equivalent to SKELETON-CODE 3.
&P4	??.??.??	Set in MDG_SKSTR4 Equivalent to SKELETON-CODE 4.
&P5	?. ??	Set in MDG_SKSTR5 Equivalent to SKELETON-CODE 5.
&P6	?xxxx	Set in MDG_SKSTR6 Equivalent to SKELETON-CODE 6.
&P7	L??.	Set in MDG_SKSTR7 Equivalent to SKELETON-CODE 7.

You can tailor the default prompts by changing the values set in SEXEC EC1060 for the above variables. You must ensure that variables MDG_SKSTR*n* (where *n* is a code in the range 2 to 7) reflect any changes you have made to the line erase character using variable MDG_DELSTR.

Because the prompts are tailorable ASG advises specifying the above parameters rather than explicitly entering the prompt in the SKELETON-TEXT clause.

The SKELETON-TEXT clause is optional. If it is omitted the input prompts are generated from the SKELETON-CODE clause.

Defining How Clauses and Keywords are Formatted by Assisted Update

To define the format in which the values of a clause or keyword are displayed in an assisted update buffer, specify:

```
EDIT-CODE-1 n1
```

where *n1* is 1 2 3 4 5 6 7 or 8

(2 is only valid for FREE-FORM-TEXT attribute types.)

To define the format in which the values of a clause or keyword updated in an assisted update buffer are filed in the repository, specify:

```
EDIT-CODE-2 n2
```

where *n2* is 1 2 3 4 5 6 7 or 8

(2 is only valid for FREE-FORM-TEXT attribute types.)

The codes specified in the EDIT-CODE-1 and EDIT-CODE-2 clauses can differ. You must select the combination of codes that are most appropriate for the clause or keyword. The effects of the different codes are as follows:

Code	EDIT-CODE-1	EDIT-CODE-2
1	Leading blank spaces deleted.	No treatment.
2	No treatment. FREE-FORM-TEXT only.	No treatment. FREE-FORM-TEXT only.
3	Leading blank spaces and commas deleted.	A single comma is entered in column 1 of the 2nd and all subsequent lines.
4	Leading blank spaces and delimiters deleted.	Delimited by single quotes ('). Any single quotes imbedded within the string are converted into double quotes ('').
5	Leading blank spaces, commas, and delimiters deleted.	Delimited by single quotes ('). Any single quotes imbedded within the string are converted into double quotes(''). A comma is entered in column 1 of the 2nd and all subsequent lines.

Code	EDIT-CODE-1	EDIT-CODE-2	
6	Values not displayed	Automatically maintained by the AUTO-REF-STRING clause of the MEMBER-TYPE containing the ATTRIBUTE-TYPE member defining the clause.	
7	Leading blank spaces deleted.	A single comma is entered in column 1 of the 2nd and all subsequent lines. A comma is not entered on a line if one of the following keywords is: a/ the first string to appear on it, or b/ the last string to appear on the preceding line. The keywords that can be specified on the preceding line and the minimum length to which they can be abbreviated are defined in SEXEC EC0960.	
7 cont.		ALIGNED	IS
		ALLOW	KNOWN-AS
		AND	LABEL
		ARE	LEFT-HAND-SIDE
		ASCENDING	LHS
		AT	MULTIVALUED-
		CASCADE	DEPENDENCY
		CONDITION-NAME	MVD
		CONSTANT	NAMED
		CONSTRAINT	NO
		CONTAINS	NOT-ALIGNED
		DATA	NOT-NULL
		DB2-COMMENT	OPTIONAL
		DB2-LABEL	OR
		DEFAULTED-AS	PARAMETERS
		DELETE	PASSING
		DESCENDING	PRIMARY
		DISALLOW	RANGE
		DUMMY	REFERENCES
		DUPLICATE	REPORTED-AS

Code	EDIT-CODE-1	EDIT-CODE-2	
		ELSE	RESTRICT
		ENTERED-AS	RHS
		ENTRY-POINT	RIGHT-HAND-
		EXPAND	SIDE
		FD	SET-NULL
		FOREIGN-KEY	SUB-DOMAIN
		FOR-BIT-DATA	TO
7 cont.		FIELDPROC	UNALIGNED
		FUNCTIONAL-	UNIQUE
		DEPENDENCY	WARN
		HELD-AS	WITH-GRANT-
		IF	OPTION
		INCLUDES	WITH-DEFAULT
		INDEXED-BY	YES
	Leading blank spaces, commas and delimiters deleted.	Leading blank spaces deleted. Delimited by single quotes (')). Any single quotes imbedded within the string are converted into double quotes ("). A single comma is entered in column 1 of the 2nd and all subsequent lines.	

You can use the environment function (provided by panel Z42200) to tailor the default delimiters. The standard string delimiter (by default ') is set in variable MDG_STADEL. The secondary string delimiter (by default ") is set in variable MDG_SECDEL.

The EDIT-CODE-1 and EDIT-CODE-2 clauses are optional. The defaults:

- EDIT-CODE-1 4 and EDIT-CODE-2 4 for CHARACTER-STRING, DATE, NAME, TEXT, and TIME attribute types.
- EDIT-CODE-1 1 and EDIT-CODE-2 1 for DECIMAL-NUMBER, INTEGER, and KEYWORD attribute types.
- EDIT-CODE-1 2 and EDIT-CODE-2 2 for FREE-FORM-TEXT attribute types.

Taking a User Exit Defining how Clauses and Keywords are Formatted by Assisted Update

To take a user exit defining the format in which the values of a clause or keyword are displayed in an assisted update buffer, specify:

```
EDIT-EXEC-1 member-name
```

where *member-name*:

- Is the name of an EXECUTIVE member on the MP-AID
- Must be delimited when entered via the command interface

To take a user exit defining the format in which the values of a clause or keyword updated in an assisted update buffer are filed in the repository, specify:

```
EDIT-EXEC-2 member-name
```

The user exit routines are defined in SEXEC members which are constructed onto the MP-AID as the specified EXECUTIVEs.

The EDIT-EXEC-1 and EDIT-EXEC-2 clauses are optional. If they are specified they will override the update and file formats defined in the EDIT-CODE-1 and EDIT-CODE-2 clauses.

Refer to Chapter 7, "User Exits," on page 171 for details of how to define a user exit routine.

Displaying Repeating Clauses and Keywords in Assisted Update

Particular clauses and keywords can be entered repeatedly in a member definition. For example, an ITEM member can have up to 15 versions of the ENTERED-AS, HELD-AS, and REPORTED-AS clauses.

To display both:

- Existing versions of a repeating clause or keyword
- An identifying keyword and input prompt enabling you to enter an additional version of the clause each time a member is updated via an assisted update buffer, specify:

  ```
  REPEAT-CODE M
  ```

To display a non-repeating clause or keyword in an assisted update buffer, specify:

```
REPEAT-CODE S
```

If you specify `S` then only one version of the clause or keyword is displayed in an assisted update buffer. The identifying keyword is followed by either an existing value or a prompt if there is no existing value. The value or prompt can be overkeyed.

The REPEAT-CODE clause is optional. REPEAT-CODE S is the default.

Defining When Clauses and Keywords are Displayed in an Assisted Update Buffer

To define when and how a clause or keyword displays in an assisted update buffer, specify:

`PROMPT-CODE` *`code`*

where *`code`* is one of these:

Code	Clause
I	The clause or keyword is always displayed.
K	The clause or keyword is only displayed if it is already present in the updated member's definition. It cannot therefore be initially entered in a member's definition via an assisted update buffer. If it is entered in the definition via some means other than assisted update (via an ADD or UPDATE command for example) it is subsequently displayed.
M	The clause or keyword is always displayed and its identifier is always filed in an updated member's definition. If you do not specify a value to follow the identifier then the filed member will fail to encode. You can also support mandatory clauses and keywords in an assisted update buffer by specifying a SKELETON-CODE 6 clause.
N	The clause or keyword is displayed when a member is first created. If it is updated when the member is created then it is also displayed during subsequent updates. If it is not updated then it is not displayed. If it is subsequently entered in the definition via some means other than assisted update (via an ADD or UPDATE command for example) then it is displayed.
O	The clause or keyword is always displayed. The clause or keyword's identifier is prefixed with the line erase characters (by default ??.). The identifier will be deleted unless you delete the line erase characters. You must allow the identifier to be deleted for unidentified keywords where only the value can be filed in a member's definition. For clauses and identified keywords that require both an identifier and a value you must delete the line erase characters, or otherwise the member will fail to encode.
P	The clause or keyword is displayed when a member is first created in the same manner as for PROMPT-CODE N. The clause or keyword's identifier is prefixed with the line erase characters in the same manner as for PROMPT-CODE O.

The PROMPT-CODE clause is optional. The default is PROMPT-CODE I.

Documenting Help for a Clause or Keyword

To document help for a clause or keyword, specify:

```
HELP text
```

where *text*:

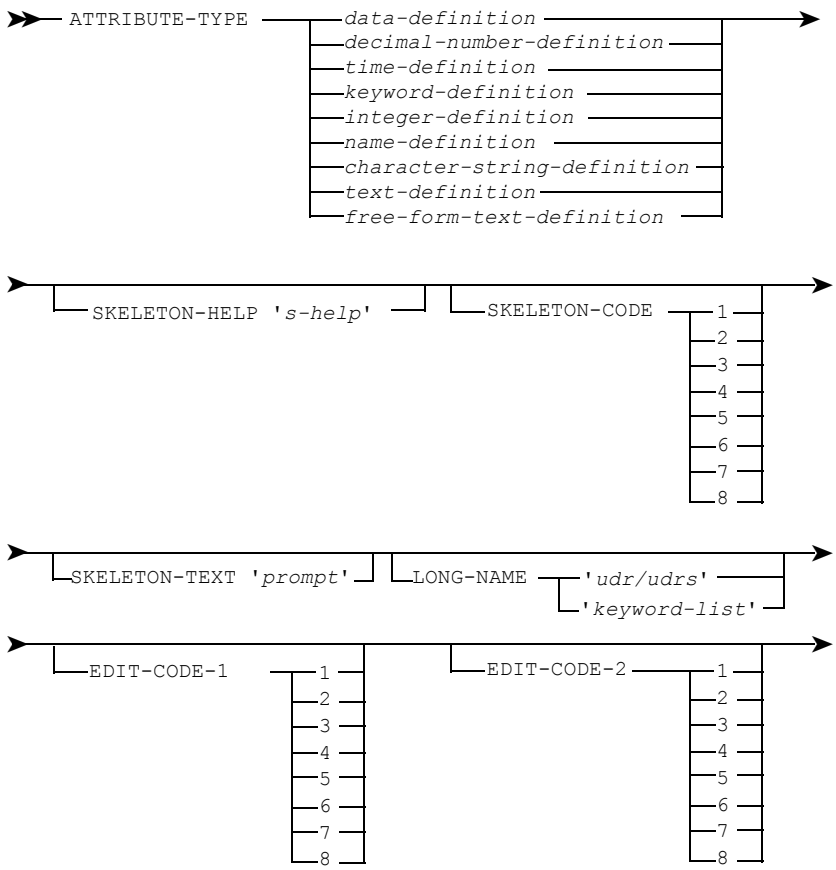
- Is free form text documenting the clause or keyword
- Is displayed as in-context help from within an assisted update buffer and in response to an MTHELP command.

Help on the member type containing the clause or keyword should be defined in the HELP clause of the relevant MEMBER-TYPE member.

The HELP clause must be the last clause specified in an ATTRIBUTE-TYPE member definition.

The HELP clause is optional.

ATTRIBUTE-TYPE Syntax



```
──┬──────────────────────────┬──┬──────────────────────────┬──────────────────────────►
  └EDIT-EXEC-1 'executive'───┘  └EDIT-EXEC-2 'executive'───┘

──┬──────────────────────────┬──┬──────────────────────────┬──────────────────────────►
  └REPEAT-CODE ──┬─S─┬──┬──┬──┘  └PROMPT-CODE ──┬─I─┬──┬──┬──┘
                 └─M─┘  │  │                    ├─K─┤  │  │
                        │  │                    ├─M─┤  │  │
                        │  │                    ├─N─┤  │  │
                        │  │                    ├─O─┤  │  │
                        │  │                    └─P─┘  │  │
──┬──────────────────────┬──┬──────────────────────┬──────────────────────────────────►
  └common-clauses────────┘  └HELP help─────────────┘

──┬─;─┬──────────────────────────────────────────────────────────────────────────────►◄
  └─.─┘
```

where:

date-definition is

```
──► DATE identifier ──┬<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<►
                      │──INDEXED-BY ──┬─PRESENCE──┬──────────────────────────────────────
                      │               └─VALUE─────┘
                      │──VALUES 'date' ──┬──────────────────────────────────────────────
                      │                  │ < , <
                      │──NORMALIZED-VALUES ndate──┘
                      │──MINIMUM-VALUES 'date' ──┬──────────────────────────────────────
                      │                          │ < ,<
                      │──NORMALIZED-MINIMUM-VALUE ndate──┘
                      │──MAXIMUM-VALUES 'date' ──┬──────────────────────────────────────
                      │                          │ < ,<
                      │──NORMALIZED-MAXIMUM-VALUE ndate──┘
                      └──multivalues──────────────────────────────────────────────────────
```

where:

identifier is

```
                  <<<<< , <<<<
──► IDENTIFIED-BY user-keyword──────────────────────────────────────────────────────────►
```

user-keyword is up to 32 characters from the standard character set for names but must not begin with an underscore.

date is a date in the format defined in the DCUST installation macro. The default is:

```
➤──day ──┬─month─┬──year ──────────────────────────➤
         ├─JAN───┤
         ├─FEB───┤
         ├─MAR───┤
         ├─APR───┤
         ├─MAY───┤
         ├─JUN───┤
         ├─JUL───┤
         ├─AUG───┤
         ├─SEP───┤
         ├─OCT───┤
         ├─NOV───┤
         └─DEC───┘
```

day is one or two numeric characters in the range 1 or 01 to 31.

month is one or two numeric characters in the range 1 or 01 to 12.

year is two or four numeric characters in the range 00 to 99 or 1000 to 2999, specifying the last two or all four characters of the year.

ndate is a date in the format yyyyddd.

yyyy is four numeric characters specifying the year and *ddd* is three numeric characters (in the range 001 to 365) specifying the number of the day within the year.

multivalues is

```
➤── MULTIPLE-VALUES ──┬──────────────────────────┬──➤
                      ├─MINIMUM-NUMBER n4 ───────┤
                      └─MAXIMUM-NUMBER n4 ───────┘
```

n4 is an integer in the range 1 through 32767.

decimal-number-definition is

```
➤──DECIMAL-NUMBER identifier──<<
```

time-definition is:

```
➤──TIME identifier ──┬──────────────────────────────────────────────────┬──➤
                     <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
                     │          < , <                                   │
                     ├─┬─VALUES 'time' ──────────────────────────────┬──┤
                     │ │                                             │  │
                     │ │                          < , <              │  │
                     │ └─NORMALIZED-VALUES ntime ────────────────────┘  │
                     │                        < , <                     │
                     ├─┬─MINIMUM-VALUE 'time' ───────────────────────┬──┤
                     │ │                                             │  │
                     │ │                                   < , <     │  │
                     │ └─NORMALIZED-MINIMUM-VALUE ntime ─────────────┘  │
                     │                        < , <                     │
                     ├─┬─MAXIMUM-VALUE 'time' ───────────────────────┬──┤
                     │ │                                             │  │
                     │ │                                   < , <     │  │
                     │ └─NORMALIZED-MAXIMUM-VALUE ntime ─────────────┘  │
                     └──multivalues ────────────────────────────────────┘
```

time is a time in the format defined in the DCUST installation macro. The default is:

```
➤──── hour minute second ──────────────────────────────────────────────➤
```

hour may be one or two numeric characters in the range 0 or 00 to 24.

minute may be one or two numeric characters in the range 0 or 00 to 59.

second may be one or two numeric characters in the range 0 or 00 to 59. The inclusion of seconds is optional.

ntime is a time in the format *hhmmss*.

hhmmss is six numeric characters in the range 000000 to 240000.

where:

hh is hour.

mm is the minute.

ss is the seconds.

identifier and *multivalues* are defined above.

keyword-definition is:

```
➤──── KEYWORD ──┬──identifier ──┬──────────────────────────────────────➤
                │               <<<<<<<<<<<<<<<  <<<<  ,<<<<
                └──NAMED keyword┘  │ ┌──index ──────┐
                                   └─┴──multivalues ┘

➤──── VALUES name ─────────────────────────────────────────────────────➤
```

keyword is a string of up to 32 characters from the standard character set for names but must not begin with an underscore

index is

```
➤──INDEXED-BY──┬─PRESENCE──┬──────────────────────────────➤
               └─VALUE─────┘
```

name is a string of up to 32 characters from the standard character set for names but must not begin with an underscore.

identifier and *multivalues* are defined above.

integer-definition is

```
                          <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
➤──INTEGER identifier ──┬──────────────────────────────┬──➤
                        │──index──────────────────────│
                        │          < , <               │
                        │──VALUES value───────────────│
                        │──MINIMUM-VALUE value────────│
                        │──MAXIMUM-VALUE value────────│
                        │──MINIMUM-LENGTH n1──────────│
                        │──MAXIMUM-LENGTH n1──────────│
                        │──multivalues────────────────│
```

value is an integer of up to 18 digits, optionally preceded by a sign character.

identifier, *index*, *n1*, and *multivalues* are defined above.

name-definition is

```
                          <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
➤──NAME identifier ─────┬──────────────────────────────┬──➤
                        │──index──────────────────────│
                        │          < , <               │
                        │──VALUES string──────────────│
                        │──MINIMUM-VALUE string───────│
                        │──MAXIMUM-VALUE string───────│
                        │──MINIMUM-LENGTH n2──────────│
                        │──MAXIMUM-LENGTH n2──────────│
                        │──multivalues────────────────│
```

string is a sting that obeys the rules for repository member names

n2 is an integer not greater than 32.

identifier, *index*, and *multivalues* are defined above.

character-string-definition is

```
➤──CHARACTER-STRING identifier──┬──<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<──┬──────────────
```

prompt is one or more text strings within which you can specify the following parameters: &P1, &P2, &P3, &P4, &P5, &P6, &P7.

executive is the name of an EXECUTIVE member on the MP-AID.

udr/udrs is a renamed UDR or UDRS clause name.

keyword-list is two or more keywords each separated by a blank space.

help is one or more lines of free form text.

common-clauses are any of the clauses common to all member types.

Note:

The commas and delimiters shown in the above syntax are required when creating an ATTRIBUTE-TYPE member via the command interface.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of the common clauses.

Refer to *ASG-ControlManager User's Guide* for details of the character set for member names.

Refer to *ASG-Manager Products Procedures Language* for details of the EXECUTIVE-ROUTINE member type.

FMT-SCREEN

The FMT-SCREEN member type defines a formatted panel for the user interface.

Refer to "FMT-SCREEN Syntax" on page 264 for the syntax of the FMT-SCREEN definition.

To define the name of a FMT-SCREEN member in the repository, enter:

SC-*name*

where *name* is an alphanumeric string of no more than 29 characters. The name of a FMT-SCREEN member must start with the prefix SC-. The length of the FMT-SCREEN name, including its prefix, must not exceed 32 characters.

Four types of panels (menus, list, input, or output panels) can be defined by the FMT-SCREEN member. The panel type is defined in the TYPE clause of the FMT-SCREEN member definition (refer to "Defining the Panel Type" on page 250).

The help for a panel is generated from members specified in the SEE clause of the FMT-SCREEN member (refer "Defining the Help for the Panel" on page 248).

Field control characters specify the types (protected or unprotected) and positions of fields in the CONTENTS clause of the FMT-SCREEN member. All field control characters exist by default but can be changed in clauses such as PROTECTED-HIGHLIGHT or UNPROTECTED-NORMAL (refer to "Defining Field Control Characters" on page 251).

All fields used by a panel can be defined either:

- In ITEM members that are related to the FMT-SCREEN member via the OUTPUTS and INPUTS clause
- In the DECLARE-FIELDS clause of the FMT-SCREEN member (refer to "Defining Input and Output Fields in the FMT-SCREEN Member" on page 254 and "Specifying a Relationship to ITEM members Defining Input Fields" on page 255).

The processing of the panel is defined by the CALLS clause (refer to "Defining the Processing of the Panel" on page 255).

The layout of the panel containing static text, input/output fields is defined in the CONTENTS clause (see "Defining the Layout of the Panel" on page 263).

Defining the Help for the Panel

To define extended help generated from the definition of an INFOBANK-PANEL or an FMT-SCREEN member, enter:

```
SEE member-name1 FOR HELP
```

where *member-name1* is the name of an INFOBANK-PANEL or FMT-SCREEN member that contains the help text.

The generated extended help is called using a function key (PF1 by default).

To define extended help generated from the definition of several members of any type except INFOBANK-PANEL and FMT-SCREEN, enter:

```
SEE member-name2 FOR HELP
```

where *member-name2* is the name of any member type other than INFOBANK-PANEL or FMT-SCREEN.

If the help text is generated from more than one member, each member must be specified on a separate line, for example:

```
SEE member-name1 FOR HELP
    member-name2 FOR HELP
    member-name3 FOR HELP
```

One extended help is generated from the specified members. The global variable MDG_MMR_CX_HELP_TYPE defines for each member type from which clause of its definition the help text is to be taken from. By default the help is generated from the HELP clause. To change the default, specify the names of the relevant clauses in the global variable MDG_MMR_CX_HELP_TYPE in the Administration Repository.

The extended help is called using a function key (PF1 by default).

To define contextual help composed from the definition of several members of any type other than INFOBANK-PANEL or FMT-SCREEN, enter:

```
SEE member-name2 FOR HELP-ID field-name
```

where:

member-name2 is the name of any member type other than INFOBANK-PANEL or FMT-SCREEN.

field-name is the name of a field, defined in the DECLARE-FIELDS clause or specified via the OUTPUTS and INPUTS clauses of the FMT-SCREEN member definition. Each member from which a help text is to be taken and each field for which help is to be generated must be specified on a separate line, for example:

```
SEE member-name1 FOR HELP-ID field-name1
    member-name2 FOR HELP-ID field-name2
    member-name3 FOR HELP-ID field-name3
```

The global variable MDG_MMR_CX_HELP_TYPE defines for each member type from which clause of its definition the help text is to be taken from. By default the help is generated from the HELP clause. To change the default, specify the names of the relevant clauses in the global variable MDG_MMR_CX_HELP_TYPE in the Administration Repository.

The contextual help is called using the character(s) defined in the HELP-IDENTIFIER clause (a question mark (?) by default).

Contextual help for a FMT-SCREEN member of the type INPUT can also be defined in the CALLS clause. Refer to "Defining the Processing of a Contextual help in FMT-SCREEN Members of the Type INPUT" on page 260 for further information.

Defining an MP-AID Name for a FMT-SCREEN Member

To define an MP-AID name for a FMT-SCREEN member, enter:

```
MPAID-NAME name
```

where *name* is a maximum of seven characters, beginning with an alphabetic character and otherwise complying with the Manager Products naming standards. The first three characters of the MP-AID name are automatically generated.

If the FMT-SCREEN member name is more than ten characters long, you must specify an MP-AID name for the UDS table.

Defining the Panel Type

To define the type of panel to be created, enter:

```
TYPE panel-type
```

where *panel-type* is one of these available types:

- MENU defines a menu from which options can be selected
- LIST defines a list panel, for instance displaying member types or members, which can be selected or processed using selection characters defined in the CALLS clause of the FMT-SCREEN member
- INPUT defines an input panel, composed of fields which have to be filled in by the user
- OUTPUT defines an output panel

Defining a Point of Return for the Control Program

To specify whether or not the current panel is to be a point of return for the control program when the user exits another panel (by default using PF3), enter:

```
APPLICATION-POINT keyword1
```

where *keyword1* is either YES or NO.

YES. Defines a panel as a point of return for the control program. If the user exits another panel at a different point in the interface the program returns to the next panel for which an APPLICATION-POINT YES has been defined. Several panels of the interface can be defined as points of return for the control program; for example:

```
Level   Panel     APPLICATION-POINT Defined

1       V00000    NO
2       X00000    YES
3       D00000    YES
4       D30000    NO
5       TD33000   NO
```

In the above example, if the user exits panel TD33000, the program returns to panel D00000 first. If the user then exits panel D00000, the program returns to X00000 and finally to V00000. The entry menu to LifeCycle Services, V00000, is always the point of return if no other application points have been defined.

NO. Suppresses the definition of an application point. NO is the default setting.

Defining the Appearance of the Panel When Returned to From Another Panel

To define the appearance of the panel when returned to from another panel, enter:

```
OPTION keyword2
```

where *keyword2* is one of the following:

- REUSE displays the panel as it was originally
- NEW refreshes the panel. All previous entries are deleted
- HOLD displays the panel as it was originally. The global variables, containing the entries of the corresponding fields, must be released by an EXECUTIVE via the CALLS clause in the FMT-SCREEN member definition. Otherwise the contents of the global variables will be held until the end of the session

Defining Field Control Characters

To define the control character(s) for protected output fields that are highlighted, enter:

```
PROTECTED-HIGHLIGHT string2
```

To define the control character(s) for unprotected input fields that are highlighted, enter:

```
UNPROTECTED-HIGHLIGHT string2
```

To define the control character(s) for protected output fields that are displayed with normal intensity, enter:

```
PROTECTED-NORMAL string2
```

To define the control character(s) for unprotected input fields that are displayed with normal intensity, enter:

```
UNPROTECTED-NORMAL string2
```

To define the control character(s) for unprotected input fields that can receive hidden text, enter:

```
UNPROTECTED-DARK string2
```

To define the control character(s) for output fields, enter:

```
FIELD-MARK string2
```

The FIELD-MARK character(s) must be used in conjunction with the other field control characters to indicate the start position of an output field.

Examples

??XP??FF indicates the start position of a protected output field displayed with normal intensity.

??HP??FF indicates the start position of a protected output field that is highlighted.

The display of the output starts at the position of the second question mark (?) in the string (??HP??FF).

To define the control character(s) that indicate the repetition of fields, enter:

```
LIST-BODY string2
```

The LIST-BODY characters indicate that fields specified in the following line are to be repeated. LIST-BODY characters must be used to indicate a repetition in a list panel. They can be used to indicate a repetition in an output panel.

The following example is an extract from the CONTENTS clause of a FMT-SCREEN member definition of the type LIST.

```
??XPSel.??HPMember                              Membertype
??BO
??HU_    ??XU                                   ??XP??FF
```

In a list panel the number of repetitions is determined by the global variable that defines the first input field following the LIST-BODY characters. In the example above the position of this variable is indicated by the field control characters ??HU.

The following is an extract from the CONTENTS clause of a FMT-SCREEN member definition of the type OUTPUT:

```
??XP     Snapshot     ??HPDisplay Condition of Current Workbench
??XP
??BO
??XP??FF
```

In an output panel the number of repetitions is specified in a global variable that defines the first field following the LIST-BODY character(s). In the above example, the position of this variable is indicated by the field control characters ??XP??FF.

To define the control character(s) to call contextual help, enter:

`HELP-IDENTIFIER` *`string2`*

The specified character(s) can be entered by the user in an input field to call its related help.

where *`string2`* is a maximum of four characters. Alphabetic and special characters can be used.

If the above clauses have not been defined, the following strings will be used by default:

Clause	Default String
PROTECTED-HIGHLIGHT	??HP
UNPROTECTED-HIGHLIGHT	??HU
PROTECTED-NORMAL	??XP
UNPROTECTED-NORMAL	??XU
UNPROTECTED-DARK	??DU
FIELD-MARK	??FF
LIST-BODY	??BO
HELP-IDENTIFIER	?

Defining Input and Output Fields in the FMT-SCREEN Member

To define an input field directly in the FMT-SCREEN member definition, enter:

```
DECLARE-FIELDS
IN variable-name1 type length mode
```

To define an output field directly in the FMT-SCREEN member definition, enter:

```
DECLARE-FIELDS
OUT variable-name1 type length mode
```

where:

variable-name1 is the name of a global variable that defines one field of the panel. The variable is defined in the FMT-SCREEN member by which it is used. ASG supplied global variables have the prefix MDG_. Please use another naming convention for user defined global variables.

type is one of these keywords:

- ALPHANUMERIC
- NUMERIC

length is an integer specifying the maximum length of the variable.

mode is one of the following:

- LOWER translate contents to lower case
- UPPER translate contents to upper case
- MIXED accept contents entered in mixed case

If mode is not defined, the UPPER keyword will be used by default.

You must define all global variables that are used by a FMT-SCREEN member defining a panel in the DECLARE-FIELDS clause, if the global variables have not been defined in ITEM members. If the DECLARE-FIELDS clause has been defined, entries in the INPUTS and OUTPUTS clauses will be ignored.

The sequence of the global variables in the DECLARE-FIELDS clause must correspond with the sequence of their field control characters in the CONTENTS clause.

Specifying a Relationship to ITEM Members Defining Output Fields

To specify a relationship to an ITEM member that defines an output field of the panel, enter:

```
OUTPUTS variable-name2
```

where *variable-name2* is the name of a global variable that defines an output field of the panel. The variable itself is defined in an ITEM member.

If the output fields of the panel have been defined in the DECLARE-FIELDS clause, the OUTPUTS clause will be ignored.

Specifying a Relationship to ITEM members Defining Input Fields

To specify a relationship to an ITEM member that defines an input field of the panel, enter:

```
INPUTS variable-name2
```

where *variable-name2* is the name of a global variable that defines an input field of the panel. The variable itself is defined in an ITEM member.

If the input fields of the panel have been defined in the DECLARE-FIELDS clause, the INPUTS clause will be ignored.

Defining the Processing of the Panel

General Definition

To define the processing of the panel, enter:

```
CALLS exec AT clause
```

where:

exec is the MP-AID name or the repository name of an SEXEC member containing Manager Products commands and/or procedures language.

`clause` is one of these keywords:

- PREINIT indicates that the specified exec is to be carried out at initialization time, even when panels are not displayed during direct path selections.
- INIT indicates that the specified exec is to be carried out at initialization time of the panel. It is not executed if the panel is not displayed during direct path selections.
- BEFORE indicates that the specified exec is to be carried out before the panel is sent/ displayed.
- AFTER indicates that the specified exec is to be carried out immediately after receiving the panel.
- PROCESS indicates that the specified exec is to be carried out when the user presses the ENTER key. PROCESS can only be used for FMT-SCREEN members of the type INPUT and LIST. If PROCESS has been specified, COMBIN and SELECT must not be used
- COMBIN defines a valid input combination for a FMT-SCREEN member of the type INPUT that must match with the user's input if the specified exec is to be carried out
- SELECT defines a selection string that initiates the specified exec in a FMT-SCREEN member of the type LIST when the string is entered before the relevant object displayed in the list.
- EXIT indicates that the specified exec is to be finished, the application cycle closed and control returned to the next application point.
- CANCEL indicates that the specified exec is to be finished and control returned to the previous panel. If a FMT-SCREEN member has been defined with OPTION HOLD the exec specified with the EXIT or the CANCEL clause must release the global variables, otherwise their contents will be held until the end of the session.
- HELP defines contextual help for a FMT-SCREEN member of the type INPUT that is called by the specified exec when the user enters the defined HELP-IDENTIFIER string in the relevant input field of the panel. All input fields used by the input panel must have been defined as global variables in ITEM members. The repository names of the relevant ITEM members must have been specified in the INPUTS clause of the FMT-SCREEN member definition.

Optionally, all clauses can be followed by the PASSING keyword, defining a field or a parameter-string that is to be processed by the called exec. If the HELP keyword has been specified, PASSING must be specified in conjunction with the HELP keyword.

To define that a parameter is to be passed on to the called exec for further processing, enter:

```
CALLS exec AT INIT PASSING string3
```

where:

exec is the MP-AID name or the repository name of an SEXEC member. This exec defines the processing of the specified string.

string3 defines a parameter consisting of alphanumeric and/or special characters. This parameter will be passed on to the specified exec for processing.

Special Definitions

Defining Error Handling

To define error handling in FMT-SCREEN members of different types, enter: `CALLS exec AT keyword3 MESSAGE nnnnn x`

where:

exec is the MP-AID name or the repository name of an SEXEC member containing MANAGER Products commands and/or procedures language. This exec writes the specified message to the message area if the condition applies.

keyword3 specifies an error condition. These conditions can be specified:

- NOINPUT specifies a condition where Enter is pressed although no entries have been input.
- NOTFOUND specifies a condition where an invalid selection is made. Do not use NOTFOUND in FMT-SCREEN members of the type OUTPUT.
- NOTPGM specifies a condition where an option is selected that has not been defined by the Systems Administrator or where an entry is made in a field for which no EXECUTIVE exists on the MP-AID. Use NOTPGM only for FMT-SCREEN members of the type MENU and INPUT.
- NOCOMBIN specifies a condition where entries are made in an invalid combination. Use NOCOMBIN only for FMT-SCREEN members of the type INPUT.
- NOLINES specifies a condition where no output can be generated from entries made in a panel. Use NOLINES only for FMT-SCREEN members of the type INPUT.
- NONUMERIC specifies a condition where a non-numeric entry is made in a field that has been defined as numeric. Use NONUMERIC only for FMT-SCREEN members of the type INPUT and LIST.
- NOHELP specifies a condition where help is called for a panel for which no help has been defined.
- NOSELECT specifies a condition where an invalid string that is not defined in the CALLS clause of the FMT-SCREEN member has been entered to select an object. NOSELECT can only be used for FMT-SCREEN members of the type LIST.

nnnnn is a five-digit integer defining the message number.

x is a one character alphabetic suffix, which denotes the type of output message. For details of messages, refer to *ASG-Manager Products Messages Guide*.

To output ASG supplied messages for user defined panels, use the SEXEC member MPEAN0000.

These messages are available:

Keyword	Message Number	Text
NOINPUT	DM43023I	PLEASE ENTER YOUR INPUT
NOTFOUND	DM43001E	INVALID SELECTION
NOTPGM	DM48000E	OPTION NOT YET DEFINED BY YOUR ADMINISTRATOR
	DM00110E	MPAID MEMBER NOT PRESENT
NOCOMBIN	DM43080E	INVALID INPUT COMBINATION, USE COMMAND ?
NOLINES	DM43206W	NO OUTPUT CREATED
NONUMERIC	DM00127E	INVALID INTEGER
NOHELP	DM44031E	NO HELP AVAILABLE
NOSELECT	DM43001E	INVALID SELECTION

Example:

To output a message if entries have been made in an invalid combination in an input panel, enter:

```
CALLS MPEAN0000 AT NOCOMBIN MESSAGE 43080 E
```

To change the output, you must define new messages for the relevant keywords. You must also define a new SEXEC member that replaces MPEAN0000 and writes the specified messages to the message area.

Defining the Processing of Selectable Options in FMT-SCREEN Members of the Type MENU

To define the processing of selectable options in FMT-SCREEN members of the type MENU, enter:

```
CALLS panel AT option-string
```

where:

panel is the repository name of a FMT-SCREEN member of any type. This FMT-SCREEN member will be called if the user enters the specified option string in the menu.

option-string is a string as defined in the CONTENTS clause of the current FMT-SCREEN member.

For each selectable option of a menu, its corresponding panel and its individual string must be defined in the CALLS clause.

Example:

```
CALLS SC-TD10000 AT 1
      SC-D20000 AT 2
      SC-D30000 AT 3
      SC-D40000 AT 4
```

Defining the Processing of Selectable Objects in FMT-SCREEN Members of the Type LIST

To define the processing of objects selected with the default character S in FMT-SCREEN members of the type LIST, enter:

```
CALLS exec AT SELECT
```

To define that the user is automatically returned to the previous panel after having selected an object with the default character S in the current panel, enter:

```
CALLS exec AT SELECT RETURN
```

To define the processing of objects selected with a user-defined selection string in FMT-SCREEN members of the type LIST, enter:

```
CALLS exec AT SELECT selection-string
```

To define that the user is automatically returned to the previous panel after having selected an object with a user-defined selection string in the current panel, enter:

CALLS *exec* AT SELECT *selection-string* RETURN

where:

exec is the MP-AID name or the repository name of an SEXEC member. This exec defines the processing of selectable objects in a FMT-SCREEN member of the type LIST.

selection-string is a user-defined string for selecting objects from the current list panel. If no selection string is defined, the default character S is used.

Defining the Validity Check and Processing of Entries in FMT-SCREEN Members of the Type INPUT

To define the validity check and processing of entries in FMT-SCREEN members of the type INPUT, enter:

```
CALLS exec AT COMBIN string
```

where:

exec is the MP-AID name or the repository name of an SEXEC member. This exec defines the processing of entries that must have been entered in a specified input combination.

string is a combination of the following characters representing one valid input combination:

- Y mandatory input in this combination
- N no input allowed in this combination
- A optional input in this combination.

The length of the string must correspond with the number of input fields specified for the input panel. Each valid input combination and its corresponding exec must be defined in the CALLS clause.

Example:

To specify the processing of an input panel, using three defined input fields in which you can make your entries in two valid combinations, enter:

```
CALLS EX-TZ60000 AT COMBIN YNN
      EX-TZ60000 AT COMBIN NYY
```

Defining the Processing of a Contextual help in FMT-SCREEN Members of the Type INPUT

To define contextual help for a FMT-SCREEN member of the type INPUT, enter:

```
CALLS exec AT HELP PASSING field
```

where:

exec is the MP-AID name or the repository name of an SEXEC member. This exec defines the generation of contextual help from an ITEM member that defines the specified field.

field is the name of a global variable that defines an input field of the panel. The variable itself is defined in an ITEM member. ASG-supplied global variables have the prefix MDG_. Please use another naming convention for user-defined global variables.

Example:

To define contextual help for three input fields used by an input panel, enter:

```
CALLS MPEARS0003 AT HELP PASSING MDG_UTR2RU_ATTR_1
      MPEARS0003 AT HELP PASSING MDG_UTR2RU_ATTR_2
      MPEARS0003 AT HELP PASSING MDG_UTR2RU_ATTR_3
```

Defining a Command Area

By default a command area is defined for menus, list panels, input, and output panels at the top of the panel. To change the default for all panels of the same type use the global variables MDG_MMR_CX_CMD_LINE and MDG_MMR_CX_CMD_TYPE. For details of these variables, refer to Chapter 6, "Customizing the Environment," on page 103.

To change the default for one panel use the COMMAND-LINE clause of the FMT-SCREEN member defining the panel. If a command area is defined in the FMT-SCREEN member the settings of the global variables MDG_MMR_CX_CMD_LINE and MDG_MMR_CX_CMD_TYPE will be ignored for the current FMT-SCREEN member.

To define a command area for the panel, enter:

```
COMMAND-LINE keyword4
```

where:

keyword4 is one of the following:

- BOTTOM positions the command area at the bottom of the panel.
- TOP positions the command area at the top of the panel.
- LINE *nn* positions the command area on a specified line.
- NO suppresses the generation of a command area. No is the default if a command area is not defined via the global variables MDG_MMR_CX_CMD_LINE and MDG_MMR_CX_CMD_TYPE.

nn is an integer, specifying the line number of the command area on the panel.

Defining the Position of the Function Key Area

To position the function key area at the bottom of the panel, enter:

```
FUNCTION-KEY BOTTOM
```

To position the function key area on a specific line, enter:

```
FUNCTION-KEY LINE nn
```

where *nn* is an integer specifying the line number of the function key area on the panel.

Defining the Allowed User Actions for the Panel

To define the actions a user is permitted to execute from the panel, enter:

```
ACTION-KEY keyword5
```

where *keyword5* is one of these:

- ENTER start a process
- HELP call the specified help
- EXIT return to the next application point
- REFRESH refresh the panel, delete previous entries
- BACKWARD scroll backward
- FORWARD scroll forward
- CANCEL return to the previous panel

If the ACTION-KEY clause is not defined, these defaults are used:

Action	Panel Types			
	MENU	INPUT	LIST	OUTPUT
BACKWARD			*	*
CANCEL	*	*	*	*
ENTER	*	*	*	
EXIT	*	*	*	*
FORWARD			*	*
HELP	*	*	*	
REFRESH		*	*	*

Defining the Position of the Message Area

To position the message area at the bottom of the panel, enter:

```
MESSAGE BOTTOM
```

To position the message area on a specific line, enter:

```
MESSAGE LINE nn
```

where *nn* is an integer, specifying the line number of the message area on the panel.

Defining a One-line Header

To define a one-line header for the panel, enter:

```
HEADER text
```

where *text* is one line of text that can consist of several strings with a maximum length of 50 characters. Field control characters can be used in the text.

Defining the Layout of the Panel

To define the layout of the panel, use the CONTENTS clause. The layout can consist of static text, output fields, and input fields.

To protect static text, use the relevant field control characters, defined in the PROTECTED-HIGHLIGHT or in the PROTECTED-NORMAL clause. Field control characters must be used to protect static text.

To protect output fields and to specify their position on the panel, use the relevant field control characters, defined in the PROTECTED-HIGHLIGHT, PROTECTED-NORMAL, and FIELD-MARK clause.

To unprotect input fields and to specify their position on the panel, use the relevant field control characters, defined in the UNPROTECTED-HIGHLIGHT, UNPROTECTED-NORMAL, and UNPROTECTED-DARK clause.

To repeat input or output fields specified in the subsequent line, use the field control characters defined in the LIST-BODY clause.

The display of static text or specified fields always begins in the second character position of the string that defines the field control character (a question mark (?) by default).

Several fields can be specified in one line. If several fields are specified in one line you should check their defined length to make sure that they do not overlap.

Note:

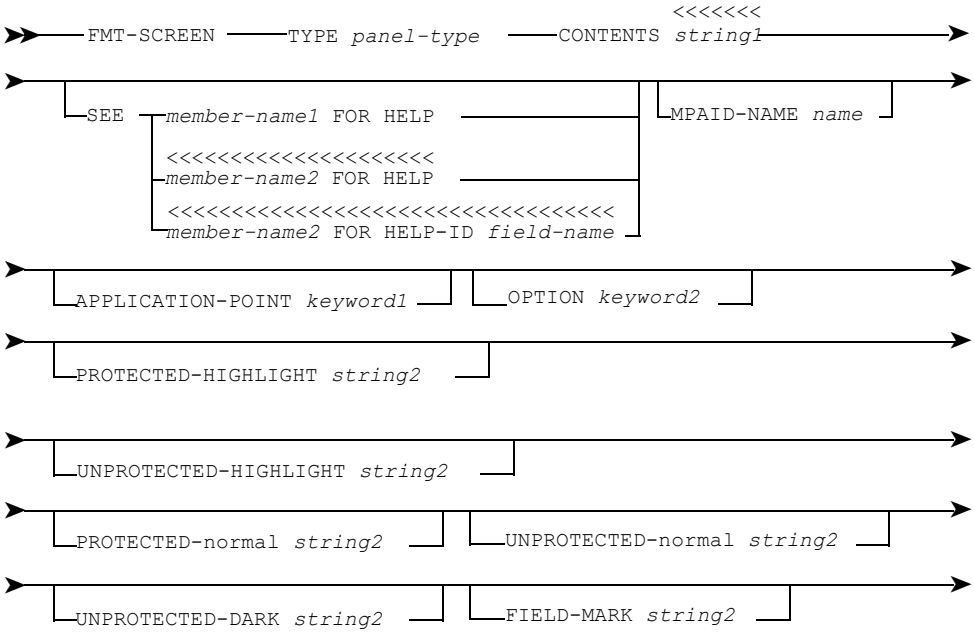
All fields specified in the CONTENTS clause by field control characters must have been defined in the DECLARE-FIELDS clause of the FMT-SCREEN member itself or in ITEM members related to the FMT-SCREEN member via the OUTPUTS and INPUTS clause.

The following example shows the definition of a menu, consisting of static text and one specified input field.

Example:

```
??XP
??XP      Admin      ??HPSystem administration??XP
??XP
??XPselect one of the following
??XP  ??HU_??HP1??XP Member Type  Select members by member type
??XP    ??HP2??XP Member           Create and update members
??XP    ??HP3??XP Enable           Enable ToolSet SERVICES
??XP    ??HPU??XP User Defined     User defined option
??XP
??XP    ??HPT??XP Tutorial         Administration tutorial
??XP
```

FMT-SCREEN Syntax



```
┌─LIST-BODY string2 ─┐ ┌─HELP-IDENTIFIER string2 ─┐
└─────────────────────┘ └─────────────────────────┘

┌─DECLARE-FIELDS ──┬─<<
```

name is a maximum of seven characters, beginning with an alphabetic character and otherwise complying with the MANAGER Products naming standards.

keyword1 is either YES or NO.

keyword2 is either REUSE, NEW, or HOLD.

string2 is a maximum of four characters. Alphabetic and special characters can be used.

variable-name1 is the name of a global variable that defines one field of the panel. The variable is defined in the DECLARE-FIELDS clause of the FMT-SCREEN member by which it is used. ASG-supplied global variables have the prefix MDG_. Please use another naming convention for user-defined global variables.

type is either ALPHANUMERIC or NUMERIC.

length is an integer specifying the maximum length of the variable.

mode is one of the following:

- LOWER translate contents to lower case
- UPPER translate contents to upper case
- MIXED accept contents entered in mixed case

If mode is not defined, the UPPER keyword is the default.

variable-name2 is the name of a global variable that defines one field of the panel. The variable itself is defined in an ITEM member. ASG-supplied global variables have the prefix MDG_. Please use another naming convention for user-defined global variables.

exec is the MP-AID name or the repository name of an SEXEC member containing Manager Products commands and/or procedures language.

panel is the repository name of a FMT-SCREEN member of any type.

clause is one of these:

- PREINIT
- INIT
- BEFORE
- AFTER
- PROCESS
- COMBIN
- SELECT
- EXIT
- CANCEL
- HELP

option-string is a string as defined in the CONTENTS clause of a FMT-SCREEN member of the type MENU.

key3 is one of these:

- NOINPUT
- NOTFOUND
- NOTPGM
- NOCOMBIN
- NOLINES
- NONUMERIC
- NOHELP
- NOSELECT

num is a five-digit integer defining the message number.

x is a one character alphabetic suffix, which denotes the type of output message.

field is the name of a global variable that defines an input field of the panel. The variable itself is defined in an ITEM member. ASG-supplied global variables have the prefix MDG_. Please use another naming convention for user-defined global variables.

string3 defines a parameter consisting of alphanumeric and/or special characters. This parameter will be passed on to the specified exec for processing.

keyword4 is one of the following:

- BOTTOM
- TOP
- NO
- LINE *nn*

nn is an integer, specifying a line number on the panel.

`keyword5` is one of these:

- ENTER Start a process.
- HELP Call the specified help.
- EXIT Return to the next application point.
- REFRESH Refresh the panel, delete previous entries.
- BACKWARD Scroll backward.
- FORWARD Scroll forward.
- CANCEL Return to the previous panel.

`text` is one line of text that can consist of several strings with a maximum length of 50 characters.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of common clauses.

HDS-TABLE

The HDS-TABLE member type specifies the options for the enable HDS tables function in ToolSet Services, which generates an HDS table and translation executive routine for ASG-ManagerView (herein called ManagerView).

Refer to "HDS-TABLE Syntax" on page 270 for the syntax of the HDS-TABLE member type.

The HDS-TABLE member type is used in conjunction with the enable HDS tables function to generate a Host Dictionary Schema table (HDS) for ManagerView.

An HDS table is a local repository representation of the syntax of mainframe repository members. It enables you to maintain host repository members locally on the programmable workstation.

The HDS-TABLE member is used to store the options for this generation process. These options include:

- The member types to be generated, and the repository information model that contains them.
- The name to be given to the generated HDS table and the translation executive routine.
- Any user-defined EA relationships to be generated.

The generation of HDS tables should be performed as part of environment enabling.

Specifying the Member Types for Generation

To specify which member types of the repository information model you wish to process, enter:

```
CONTAINS member-type-list
```

where *member-type-list* is a list of one or more MEMBER-TYPE and MEMBER-TYPE-GROUP member names. Each member name must be separated by a comma if entered via the command interface.

Specifying the Relationship Types for Generation

To specify which relationship types of the repository information model you wish to process, enter:

```
RELATIONSHIPS relationship-type-list
```

where *relationship-type-list* is a list of one or more RELATIONSHIP-TYPE and RELATIONSHIP-GROUP member names. Each member name must be separated by a comma if entered via the command interface.

Specifying a Name for the Generated HDS Table

To specify the name to be given to the generated HDS table, enter:

```
MPAID-NAME hds-table
```

where *hds-table* is a string of 8 characters or fewer.

The name is automatically prefixed with MH when the HDS table is generated.

If you do not include an MPAID-NAME clause in the HDS-TABLE member definition, the HDS table will be given the MP-AID name of the member you specified in the SEE clause, prefixed with MH.

Specifying the RIM for Generation

To specify which repository information model (RIM) contains the member types to be processed, enter:

```
SEE rim-name
```

rim-name is the name of the HIERARCHY member for the RIM.

Specifying a Name for the Generated Translation Executive Routine

To specify the name to be given to the generated translation executive routine, enter:

```
TRANSLATION-EXEC exec-name
```

exec-name is a string of 10 characters or fewer.

If this clause is omitted, the translation executive routine is given the same name as the generated HDS table.

Including User-Defined EA Relationships in the Generation

To include user-defined EA relationships in the generation process, enter:

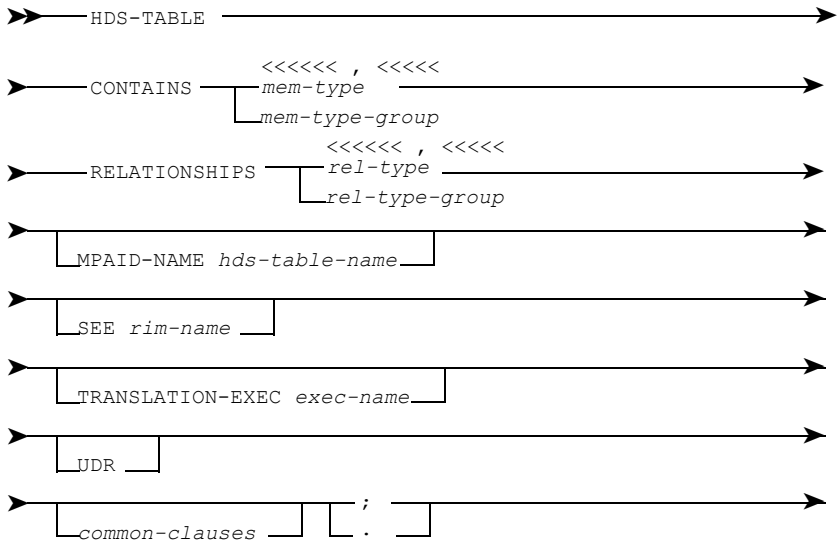
```
UDR
```

In the enable HDS tables input panel, you must specify the password of the Administration Repository, and the name and password of the repository that contains the user-defined relationships. The relationships are then automatically included in the generation.

Example

```
HDS-TABLE
CONTAINS UM-ITEM, UM-GROUP
MPAID-NAME TABLE1
SEE UH777
TRANSLATION-EXEC TR-MVW
;
```

HDS-TABLE Syntax



where:

mem-type is a MEMBER-TYPE member name.

mem-type-group is a MEMBER-TYPE-GROUP member name.

rel-type is a RELATIONSHIP-TYPE member name.

rel-type-group is a RELATIONSHIP-GROUP member name.

hds-table-name is the name to be given to the generated HDS table stored on the MP-AID.

rim-name is the name of the HIERARCHY member for the desired RIM.

exec-name is the name to be given to the generated translation executive routine on the MP-AID.

common-clauses are as defined in *ASG-Manager Products Dictionary/Repository User's Guide*.

HIERARCHY

The HIERARCHY member type specifies the member types contained in the repository information model (RIM).

Refer to "HIERARCHY Syntax" on page 279 for the syntax of the HIERARCHY member type.

Naming the MP-AID Members Generated from the RIM

To define the names of the MP-AID members generated from the RIM, specify:

```
MP-AID-NAME name
```

where *name*:

- Can be a maximum of five characters from the standard character set for names but must not begin with an underscore.
- Defines the name of the UDS-TABLE member and part of the name of the EXECUTIVE members created on the MP-AID.

The members are constructed onto the MP-AID by the enable ToolSet Services functions (provided by panel A70000 and the UX and CONSTRUCT commands).

The MP-AID-NAME clause is optional.

If you do not specify an MP-AID-NAME clause then the first five characters of the HIERARCHY member name will form both the UDS-TABLE member name and part of the EXECUTIVE member's names.

If several HIERARCHY member names share the same first five characters then duplication of names will result.

A constructed EXECUTIVE member will replace an existing EXECUTIVE member of the same name.

A constructed UDS-TABLE replaces an existing UDS-TABLE of the same name unless the existing UDS-TABLE is applied to a repository in which case you cannot construct the new table. ASG recommends specifying a unique MP-AID-NAME clause for each HIERARCHY member.

Refer to "Analyzing Generated Executives" on page 99 for details of the EXECUTIVE member names generated.

Specifying the Entity Member Types Contained in the RIM

To define the entity member types contained in the RIM, specify:

```
CONTAINS member-type-list
```

where *member-type-list* is a list of MEMBER-TYPE and MEMBER-TYPE-GROUP member names. Each member name must be separated by a comma if entered via the command interface.

If a MEMBER-TYPE-GROUP is specified, then all MEMBER-TYPEs it directly or indirectly contains are included in the RIM.

Each MEMBER-TYPE-GROUP specified in the CONTAINS clause will define an option on the member type cluster menu. MEMBER-TYPE-GROUPs indirectly contained in the HIERARCHY member will not define a cluster menu option. All MEMBER-TYPEs directly specified in the CONTAINS clause are selected from this cluster menu option:

```
N None Types Without Cluster
```

By grouping in a MEMBER-TYPE-GROUP any unrelated member types that do not fit into other groups, you can define a more meaningful menu entry.

A maximum of 13 options can be defined on a member type cluster menu.

The relationship member types contained in the RIM must be specified in the RELATIONSHIPS or ALTERNATIVE-RELATIONSHIPS clauses.

The CONTAINS clause is mandatory.

Assigning Values to Entity Member Types

To identify a user-defined entity member type by assigning it a unique numeric suffix, specify:

```
SYNONYM member-type IS BASED-ON base-member-type
SUFFIX n
```

where:

member-type is a user-defined MEMBER-TYPE member name.

base-member-type is an ASG-supplied MEMBER-TYPE member name.

n is an integer in the range 1 to 254.

Only user-defined MEMBER-TYPEs based on ASG-supplied MEMBER-TYPEs can be specified in the SYNONYM clause.

The SYNONYM clause is optional.

If you do not specify a SYNONYM clause for a user-defined MEMBER-TYPE, then a default clause is generated and inserted into the HIERARCHY member definition when the RIM is constructed onto the MP-AID.

You should take great care if you intend to manually maintain the SYNONYM clause.

For example, if you update a HIERARCHY member in order to redefine a RIM, you are recommended to leave unchanged those SYNONYM clauses that relate to member types that appear both in the old and the new RIM. A member type whose SYNONYM clause has been changed may be treated as a new member type in the redefined RIM and cause the enable ToolSet Services compare function (provided by panel A70000 and the COMPARE UDS command) to fail.

Specifying the Relationship Member Types Included in the RIM

To define the relationship member types included in the RIM, specify:

```
RELATIONSHIPS relationship-type-list
```

where *relationship-type-list* is a list of one or more RELATIONSHIP-TYPE and RELATIONSHIP-GROUP member names. Each member name must be separated by a comma if entered via the command interface.

All relationship member types defined by the specified RELATIONSHIP-TYPEs, or by the RELATIONSHIP-TYPEs directly and indirectly contained in the specified RELATIONSHIP-GROUPs, are included in the RIM.

All the RELATIONSHIP-TYPE members directly and indirectly specified in the RELATIONSHIPS clause should also be specified in the SEE clause of the MEMBER-TYPE-GROUP member that groups their source and target member types.

If a RELATIONSHIP-TYPE member is specified in an ALTERNATIVE-RELATIONSHIPS clause then it need not be specified in a RELATIONSHIPS clause.

The RELATIONSHIPS clause is optional.

Defining Mutually Exclusive Relationship Member Types

To define a set of mutually exclusive relationship member types, specify:

```
ALTERNATIVE-RELATIONSHIPS relationship-type-list
```

where *relationship-type-list* is two or more RELATIONSHIP-TYPE member names each separated by an ELSE keyword.

For example, if you specified:

```
ALTERNATIVE-RELATIONSHIPS SYSTEM-INPUTS-FILE ELSE
PROGRAM-OUTPUTS-FILE ELSE MODULE-UPDATES-FILE
```

then the same FILE member could not be the target of two relationship members having any two of the three alternative relationship member types.

Multiple sets of mutually exclusive relationship member types can be defined. The same RELATIONSHIP-TYPE member can be specified in more than one set. Each set must be separated by a comma if entered via the command interface.

Mutually exclusive RELATIONSHIP-TYPE members must share at least one common MEMBER-TYPE or RELATIONSHIP-TYPE member in their SOURCE or TARGET clause.

If a RELATIONSHIP-TYPE member has the same MEMBER-TYPE or RELATIONSHIP-TYPE as both its source and target, then you must specify the SOURCE or TARGET keywords in the ALTERNATIVE-RELATIONSHIPS clause to indicate whether it is the source or the target that is mutualy exclusive.

For example, to define that the target of PROGRAM-GENERATES-PROGRAM is mutually exclusive with the source of PROGRAM-GENERATES-REPORT, specify:

```
ALTERNATIVE-RELATIONSHIPS PROGRAM-GENERATES-PROGRAM
TARGET ELSE PROGRAM-GENERATES-REPORT
```

A PROGRAM member could not be both the target of a PROGRAM-GENERATES-PROGRAM member and the source of a PROGRAM-GENERATES-REPORT member but could be the source of both a PROGRAM-GENERATES-PROGRAM and a PROGRAM-GENERATES-REPORT member.

To define that the source of both PROGRAM-GENERATES-PROGRAM and PROGRAM-GENERATES-REPORT are mutually exclusive, specify:

```
ALTERNATIVE-RELATIONSHIPS PROGRAM-GENERATES-PROGRAM
SOURCE ELSE PROGRAM-GENERATES-REPORT
```

To define that both the source and target of PROGRAM-GENERATES-PROGRAM is mutually exclusive with the source of PROGRAM-GENERATES-REPORT, specify:

```
ALTERNATIVE-RELATIONSHIPS PROGRAM-GENERATES-PROGRAM
SOURCE ELSE PROGRAM-GENERATES-PROGRAM TARGET ELSE
PROGRAM-GENERATES-REPORT
```

All the RELATIONSHIP-TYPE members specified in the ALTERNATIVE-RELATIONSHIPS clause should also be specified in the SEE clause of the MEMBER-TYPE-GROUP member that groups their source and target member types.

If a RELATIONSHIP-TYPE member is specified in an ALTERNATIVE-RELATIONSHIPS clause then it need not be specified in a RELATIONSHIPS clause.

The ALTERNATIVE-RELATIONSHIPS clause is optional.

Assigning Values to Relationship Member Types

To identify a relationship member type by assigning it a unique numeric value, specify:

`RELATIONSHIP-VALUE` *`relationship-type n`*

where:

`relationship-type` is a RELATIONSHIP-TYPE member name.

`n` is an integer in the range 1 to 12192.

The RELATIONSHIP-VALUE clause is optional.

If you do not specify a RELATIONSHIP-VALUE clause for a RELATIONSHIP-TYPE, then a default clause is generated and inserted into the HIERARCHY member definition when the RIM is constructed onto the MP-AID.

You should take great care if you intend to manually maintain the RELATIONSHIP-VALUE clause.

For example, if you update a HIERARCHY member in order to redefine a RIM, you are recommended to leave unchanged those RELATIONSHIP-VALUE clauses that relate to relationship member types that appear both in the old and the new RIM. A member type whose RELATIONSHIP-VALUE has been changed may be treated as a new member type in the redefined RIM and cause the enable ToolSet Services compare functions (provided by panel A70000 and the COMPARE UDS command) to fail.

Defining Collective Member Types

To define a collective member type, specify:

`COLLECTIVE` *`name`* `INCLUDES` *`member-type-list`*

where:

`name`:

- Can be a maximum of 32 characters from the standard character set for names but must not begin with an underscore
- Cannot be the same as the PRIMARY-NAME of any RELATIONSHIP-TYPE member, or the interrogate keyword (specified in the INTERROGATE-KEYWORD clause or generated by default from the ENCODE-KEYWORD clause) of any MEMBER-TYPE member defining the RIM

`member-type-list` is a list of one of more MEMBER-TYPE, MEMBER-TYPE-GROUP, RELATIONSHIP-TYPE, and RELATIONSHIP-GROUP member names. Each member name must be separated by a comma if entered via the command interface.

The collective member type can be specified in any function that can be applied to a selection of members according to their member type. All members having any of the member types specified in the member-type-list will be processed.

You can specify several COLLECTIVE clauses in a HIERARCHY definition each specifying a collective member type for a separate set of member types. A member type can appear in more than one COLLECTIVE clause.

The COLLECTIVE clause is optional.

Specifying the User-defined Attributes Common to all Member Types

To define the user-defined attributes that can be included in the definition of any repository member, specify:

`COMMON-ATTRIBUTES` *`attribute-list`*

where *`attribute-list`* is:

- The names of one or more ATTRIBUTE-TYPE or ATTRIBUTE-GROUP members. All attributes defined by the named ATTRIBUTE-TYPEs or by the ATTRIBUTE-TYPEs directly and indirectly contained in the named ATTRIBUTE-GROUPs can be included in the definition of any member. Each member name must be separated by a comma if entered via the command interface.
- One or more sets of mutually exclusive ATTRIBUTE-TYPE members. Each set must contain two or more ATTRIBUTE-TYPE member names each separated by an ELSE keyword. The same ATTRIBUTE-TYPE member name can be specified in more than one set. Only one of the attributes defined by the alternative ATTRIBUTE-TYPE members can be present in the definition of a member. Each set must be separated by a comma if entered via the command interface.

Each individual ATTRIBUTE-TYPE or ATTRIBUTE-GROUP member name or each set of alternative ATTRIBUTE-TYPE members can be followed with:

OPTIONAL NO to specify that:

- An attribute or all the attributes contained in the named group
- One of the mutually exclusive attributes within a set must be present when a member is encoded.

OPTIONAL YES to specify that the attribute(s) need not be present.

OPTIONAL WARN to specify that the attribute(s) need not be present but, if not a warning message displays.

The default is OPTIONAL YES.

The COMMON-ATTRIBUTES clause is optional.

Assigning Parameter, Line, and Format Line Numbers to User-defined Attributes

To assign each user-defined attribute in the RIM a parameter number, line number and format line number, specify:

```
UDO attribute-name IS PARAMETER-NUMBER n1 LINE n2
           KEYWORD IS PARAMETER-NUMBER n3 LINE n4
                              FORMAT-LINE-NUMBER n5
```

where:

attribute-name is an ATTRIBUTE-TYPE member name.

n1, *n2*, *n3*, and *n4* are any integers in the range 1 to 32767 and *n5* is an integer which is a multiple of three in the range 2001 to 32767. The values of *n1*, *n2*, *n3*, *n4* and *n5* must be unique to the user-defined attribute.

The numbers enable you to define:

- TRANSLATION-RULE members in order to export information from a Manager Products repository
- FORMAT members in order to produce user-defined reports.

The UDO clause is optional.

If you do not specify a UDO clause for a user-defined attribute, then a default clause is generated and inserted into the HIERARCHY member definition when the RIM is constructed onto the MP-AID.

The KEYWORD clause, excluding LINE *n4*, must be included for all attributes except those having an unidentified KEYWORD attribute type. LINE *n4* can only be included for attributes having a TEXT attribute type.

A new clause is automatically generated and added to the end of the HIERARCHY definition if you specify a UDO clause that is syntactically correct but does not conform to the above rules. The numbers in the automatically generated clause are the numbers that must be used in a TRANSLATION-RULE or FORMAT member.

The numbers assigned to an attribute are output by the SHOW UDS command.

Specifying the UDR and UDRS Clauses to be Included in the RIM

To select the UDR and UDRS clauses to be included in the RIM, specify:

```
SEE attribute-type-list
```

where *attribute-type-list* is a list of ATTRIBUTE-TYPE member names. Each member name must be separated by a comma if entered via the command interface.

The specified ATTRIBUTE-TYPE members define:

- The UDR clauses (UDR1 to UDR9) providing user definable EA relationship types
- The UDRS subclause available with each UDR clause

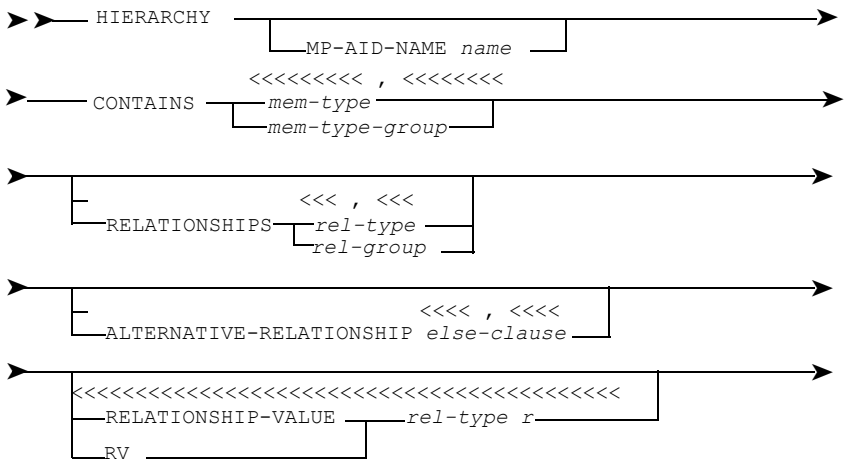
Each ATTRIBUTE-TYPE member has a LONG-NAME clause defining the new name of the clause.

The ATTRIBUTE-TYPE members defining UDR clauses must also be specified in the SEE clause of the relevant MEMBER-TYPE members in order for them to be displayed during assisted update.

The ATTRIBUTE-TYPE member defining the UDRS clause is only required in order to define an alternative clause name and must not be specified in the MEMBER-TYPE SEE clause. To display a UDRS clause during assisted update you must specify it in the SKELETON-TEXT clause of the ATTRIBUTE-TYPE members defining clauses UDR1 to UDR9.

The SEE clause is optional. All the ASG-supplied UDR and UDRS clauses are available (with their default names) if it is omitted.

HIERARCHY Syntax


HIERARCHY
MP-AID-NAME name
CONTAINS
<<<<<<<<< , <<<<<<<<
mem-type
mem-type-group
RELATIONSHIPS
<<< , <<<
rel-type
rel-group
ALTERNATIVE-RELATIONSHIP else-clause
<<<< , <<<<
<<<<<<<

```
<<
```

attribute-clause is:

```
➤──┬─attr-group──────────────────────┬──┬──────────────────────────────┬──➤
   └─attr-type──┬─<<<<<<<<<<<<<<<<<<<<─┘  ├─OPTIONAL YES─────────────────┤
                └─ELSE attr-type───────┘  ├─OPTIONAL NO──────────────────┤
                                          └─OPTIONAL WARN────────────────┘
```

where:

attr-group is an ATTRIBUTE-GROUP member name.

attr-type is an ATTRIBUTE-TYPE member name.

udr/udrs-att-type is an ATTRIBUTE-TYPE member name. The member defines a UDR clause or UDRS subclause.

s is an integer in the range 1 to 254.

parameters is:

```
➤──── IS PARAMETER-NUMBER n1 LINE n2 ──┬──────────────┬──────────────➤
                                       └─keyword──────┘
```

where:

n1 and *n2* are any integers in the range 1 to 32767.

keyword is:

```
➤──── KEYWORD IS PARAMETER-NUMBER n3 ──┬──────────────┬──────────────➤
                                       └─LINE n4──────┘
```

where:

n3 and *n4* are any integers in the range 1 to 32767.

n5 is an integer which is a multiple of three in the range 2001 to 32767.

common-clauses are any of the clauses common to all member types with the exception of the SEE clause which is used in the HIERARCHY member type for special purposes.

Note:

The commas shown in the above syntax are required when creating HIERARCHY members via the command interface.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of the common clauses. Refer to *ASG-ControlManager User's Guide* for details of the character set for names.

INFOBANK-PANEL

Refer to *ASG-Manager Products User Defined InfoSystem* for details of the INFOBANK-PANEL member type.

ITEM

The purpose of the ITEM member type is:

- Documenting the lowest levels of data, for instance data elements
- Defining global variables

Refer to "ITEM Syntax" on page 285 for the syntax of the ITEM member definition.

FMT-SCREEN members use global variables which define input and output fields of a formatted panel. An ITEM member that defines a field of a panel is therefore used as a global variable by the FMT-SCREEN member defining the panel.

The names of ASG-supplied ITEM members used as global variables start with the prefix MDG_. Please use another naming convention, for instance UDV_ for user-defined global variables.

To specify the name of an ITEM member defining a user-defined global variable in the repository, you might enter:

```
UDV_name
```

where *name* is an alphanumeric string of no more than 28 characters. The maximum length of the ITEM name, including its prefix, must not exceed 32 characters.

Defining a Title

To define a one-line title that will be used to define the HELP clause of the ITEM member, enter:

```
TITLE text1
```

where *text1* is a one-line title that can consist of several strings with a maximum length of 40 characters.

The title displays in the contextual help of the FMT-SCREEN member that uses this ITEM as a global variable in its definition.

Defining Lower, Upper or Mixed Case Mode

To specify whether the contents of the ITEM used as a global variable is to be translated to lower, upper or mixed case, enter:

```
MODE keyword
```

where *keyword* is one of the following:

- LOWER translate contents to lower case
- UPPER translate contents to upper case
- MIXED accept contents entered in mixed case

If the MODE clause is not defined, the UPPER keyword will be used by default.

Defining Valid Input Values

To define the valid values a variable can receive, enter:

```
INPUT-VALUE string
```

where *string* specifies a value whose type and length depends on the form description defined in the DEFAULTED-AS clause of the ITEM member. For instance, if you define the ITEM member as DEFAULTED-AS NUMERIC 4 only numeric strings with a maximum length of four characters can be defined as strings in the INPUT-VALUE clause.

For example, if an ITEM member is defined as DEFAULTED-AS ALPHABETIC 10, the INPUT-VALUE clause can only contain alphabetic strings with a maximum length of ten characters.

```
INPUT-VALUE AMENDED
            CURRENT
            CHANGED
            DIVERGING
            REVERIFIED
            UNVERIFIED
```

Only strings specified in the INPUT-VALUE clause are accepted when entered in the field defined by the ITEM member.

If the NOTE clause is defined in the ITEM member, the entry displays as a message.

If the user accesses the contextual help, help text will be displayed along with the valid entries for the field as defined in the INPUT-VALUE clause. From within the help, the user can select from a list of valid values which are written to the field automatically.

To define a comment that is displayed with the specified value in the help, enter:

```
INPUT-VALUE AMENDED; source record in current and base statuses
        CURRENT;  records of index-names in current status
        CHANGED;  condition of index-names changed in current status
        DIVERGING; index-names visible from current status
        REVERIFIED; encoded record in current status
        UNVERIFIED; unencoded record visible from current status
```

A semi-colon (;) must be used to indicate the end of the string and the beginning of the comment.

Defining the Form of the Data

To define the form of the data described by the ITEM member, enter:

```
DEFAULTED-AS form-description
```

Refer to *ASG-Manager Products Dictionary/Repository User's Guide*, for details of form-description.

Defining Help

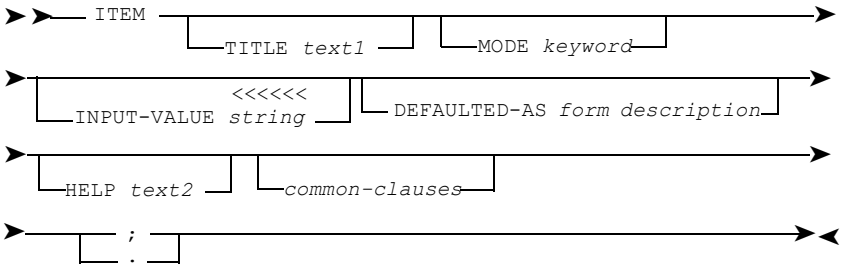
To define help text for an ITEM member, enter:

```
HELP text2
```

where *text2* consists of free-form text of unlimited length. If the ITEM member defines a global variable that is used as a field by a FMT-SCREEN member, the text should explain the purpose of the field, such as its valid entries. When the panel interface is enabled, the help for a panel is generated from the HELP clauses of the ITEM members which are used by the FMT-SCREEN member defining the panel.

To call the generated contextual help, enter the control character (defined in the HELP-IDENTIFIER clause of the FMT-SCREEN member) in the relevant field of the panel.

ITEM Syntax



where:

text1 is a one-line title that can consist of several strings with a maximum length of 40 characters.

keyword is one of the following:

- LOWER translate contents to lower case
- UPPER translate contents to upper case
- MIXED accept contents entered in mixed case

string specifies a value whose type and length depends on the form description defined in the DEFAULTED-AS clause of the ITEM member.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide*, for details of *form-description*.

text2 is free-form text of unlimited length.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of *common clauses*.

MEMBER-TYPE

MEMBER-TYPE defines an entity member type.

Refer to "MEMBER-TYPE Syntax" on page 303 for the syntax of the MEMBER-TYPE member type.

Defining a Base or User-defined Member Type

To define an ASG-supplied (base) member type or its equivalent, specify:

```
IS member-type
```

where *member-type* is a base member type.

Within one RIM different MEMBER-TYPE members cannot specify the same base member-type in their IS clause.

You can use a MEMBER-TYPE member containing an IS clause to define your own equivalent to a base member type. For example, you could rename its encode keyword in order to support your national language or the terminology used in your organization.

To define a user-defined member type, specify:

```
BASED-ON member-type
```

where *member-type* is the base member type on which the user-defined member type is modeled.

A maximum of 254 user-defined member types can be based on a single base member type.

For example, to define member types modeled on ITEM:

- Firstly, define a single MEMBER-TYPE member with an IS ITEM clause
- Subsequently, define MEMBER-TYPEs with a BASED-ON ITEM clause

The definitions of base and user-defined MEMBER-TYPE members can with the exception of the REPORT-DOWN-TO-KEYWORDS and GENERIC-ATTRIBUTES clauses contain the same clauses. The REPORT-DOWN-TO-KEYWORDS and GENERIC-ATTRIBUTES clauses can only be specified in a base member type.

You can include IMS member types in your RIM, or define your own equivalents, by specifying each in an IS clause. User-defined member types cannot be BASED-ON IMS member types.

Some ADABAS, IMS, MARKIV, and TOTAL member types are defined by two levels of base MEMBER-TYPE. The first level defines the member type as defining a database, segment, PCB or file. The second level defines the type of database, segment, PCB or file.

All second level MEMBER-TYPEs must be specified in the HIERARCHY defining the RIM but it is not essential to specify first level MEMBER-TYPEs. Default first level MEMBER-TYPEs are generated in your RIM from second level MEMBER-TYPEs if you do not specify them. If you have defined your own equivalent to a first level MEMBER-TYPE you must specify it in the HIERARCHY member. For completeness of documentation ASG recommends that all MEMBER-TYPEs be specified. First level member types must contain a LONG-NAME MMR-DISAPPEAR clause.

An IS clause or a BASED-ON clause is mandatory.

Defining the Keywords With Which the Member Type is Encoded

To define the keywords with which the member type is encoded, specify:

`ENCODE-KEYWORDS` *`keyword-list`*

where *`keyword-list`* is one or more keywords each of which can contain up to 32 characters from the standard character set for names but must not begin with an underscore.

Each keyword in the keyword-list must be separated by a comma if entered via the command interface.

The keywords must not be the same as the ENCODE-KEYWORDS of any other MEMBER-TYPE or the PRIMARY-NAME of any RELATIONSHIP-TYPE member defining the RIM.

The keywords must not be the same as the name of any ATTRIBUTE-TYPE of that MEMBER-TYPE.

If several keywords are specified, then members of that type can be encoded using any of the keywords.

Only the first keyword specified in the ENCODE-KEYWORDS clause is displayed in selection list panels and assisted update buffers regardless of which keyword is used in the definition of the member being listed or updated.

The ENCODE-KEYWORDS clause is mandatory for user-defined MEMBER-TYPEs (those containing a BASED-ON clause). ASG-supplied base MEMBER-TYPEs (those containing an IS clause) must contain either an ENCODE-KEYWORDS or a LONG-NAME clause.

Refer to "Defining a Base or User-defined Member Type" on page 286 for details of two level member types.

Refer to *ASG-ControlManager User's Guide* for details of the standard character set for names.

Defining Keywords With Which the Member Type can be Interrogated

To define keywords with which the member type can be interrogated, specify:

```
INTERROGATE-KEYWORDS keyword-list
```

where *keyword-list* is one or more keywords each of which can contain up to 32 characters from the standard character set for names but must not begin with an underscore.

Each keyword in the keyword-list must be separated by a comma if entered via the command interface.

The keywords must not be the same as:

- The PRIMARY-NAME of any RELATIONSHIP-TYPE member
- The interrogate keywords (specified in the INTERROGATE-KEYWORDS clause or generated by default from the ENCODE-KEYWORDS clause) of any other MEMBER-TYPE member
- Any collective member type specified in the COLLECTIVE clause of the HIERARCHY member defining the RIM

Any of the specified keywords can be used to interrogate the member type.

The INTERROGATE-KEYWORDS clause is optional. If it is omitted, default interrogate keywords are generated from the ENCODE-KEYWORDS clause when the RIM is constructed onto the MP-AID. The default keywords are the encode keywords followed by S, or by ES when the keyword ends in S, SH, CH or X. For example, an interrogate keyword of BATCHES would be generated from the encode keyword BATCH.

If an ENCODE-KEYWORDS clause does not exist (as in base MEMBER-TYPEs containing a LONG-NAME clause), then default interrogate keywords are not generated. ASG recommends defining an INTERROGATE-KEYWORDS clause for MEMBER-TYPEs that do not have an ENCODE-KEYWORDS clause.

The interrogate keyword for a first level member type will display all members sharing the first level. The interrogate keyword for a second level member type will only display members having that member type.

Refer to *ASG-ControlManager User's Guide* for details of the character set for names.

Refer to "Defining a Base or User-defined Member Type" on page 286 for details of the two level member types.

Defining Keywords That Can Be Specified in a REPORT DOWN-TO Command

To define keywords that can be specified in the REPORT DOWN-TO command, specify:

```
REPORT-DOWN-TO-KEYWORDS keyword-list;
```

where *keyword-list* is one or more keywords each of which can contain up to 32 characters from the standard character set for names but must not begin with an underscore.

Each keyword in the keyword-list must be separated by a comma if entered via the command interface.

The keywords must not be the same as:

- The PRIMARY-NAME of any RELATIONSHIP-TYPE member
- The interrogate keywords (specified in the INTERROGATE-KEYWORDS clause or generated by default from the ENCODE-KEYWORDS clause) of any other MEMBER-TYPE member defining the RIM.

The REPORT-DOWN-TO clause is optional and can only be specified in a base MEMBER-TYPE (containing an IS clause).

If it is omitted, only the interrogate keywords of a member type can be specified in the REPORT DOWN-TO command.

Refer to *ASG-ControlManager User's Guide* for details of the character set for names.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of the REPORT command.

Tailoring GLOSSARY, REPORT, WHAT, and WHICH Output

To define the name that identifies the member type in the output from the GLOSSARY, REPORT, WHAT, and WHICH commands (and any panel interface functions that internally execute these commands), specify:

```
STANDARD-LITERAL standard-literal
```

where *standard-literal* can be a maximum of:

- 32 characters long
- 12 characters long if you do not also define a SHORT-LITERAL clause

- 30 characters long if you do not also define a PLURAL-LITERAL clause and the last character of the standard literal is CH, S, SH, or X
- 31 characters long if you do not also define a PLURAL-LITERAL clause and the last character of the standard literal is any character other than CH, S, SH, or X.
- Must be delimited if entered via the command interface

Standard literals defined in first level MEMBER-TYPEs are not displayed in interrogation output.

The STANDARD-LITERAL clause is mandatory.

Refer to "Defining a Base or User-defined Member Type" on page 286 for details of the two level MEMBER-TYPEs.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of the GLOSSARY, REPORT, WHAT, and WHICH commands.

Tailoring LIST Output

To define the name that identifies the member type in the output from the LIST command (and any panel interface functions that internally execute this command), specify:

`SHORT-LITERAL` *`short-literal`*

where *`short-literal`*:

- Can be a maximum of 12 characters long
- Must be delimited if entered via the command interface

Short literals defined in first level MEMBER-TYPEs are not displayed in interrogation output.

The SHORT-LITERAL clause is optional.

If you do not specify a SHORT-LITERAL clause then the STANDARD-LITERAL clause is taken as the default. If the standard literal is more than 12 characters long, the default is rejected when the RIM is constructed onto the MP-AID.

Refer to "Defining a Base or User-defined Member Type" on page 286 for details of the two level MEMBER-TYPEs.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of the LIST command.

Tailoring SHOW UDS Output

To define the name that identifies member types defined by two levels of MEMBER-TYPE member (for example IMS member types) in message and SHOW command output, specify:

`LONG-LITERAL long-literal`

where *long-literal*:

- Is a maximum of 32 characters long
- Must be delimited if entered via the command interface

If you define a LONG-LITERAL clause for a member type that is not defined in two levels of MEMBER-TYPE member, then it is ignored when the RIM is constructed.

Long literals defined in first level MEMBER-TYPEs are not displayed in output.

The LONG-LITERAL clause is optional.

If you do not specify a LONG-LITERAL clause then the STANDARD-LITERAL clause is taken as the default.

Refer to "Defining a Base or User-defined Member Type" on page 286 for details of the two level MEMBER-TYPEs.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of the SHOW UDS command.

Tailoring GLOSSARY and LIST Headings and Totals Output

To define the name that identifies the member type in the headings and totals output of the LIST and GLOSSARY commands (and any panel interface functions that internally execute these commands), specify:

`PLURAL-LITERAL plural-literal`

where *plural-literal*:

- Can be a maximum of 32 characters long
- Must be delimited if entered via the command interface

The PLURAL-LITERAL clause is optional.

If you do not specify a PLURAL-LITERAL clause then the default is taken from the STANDARD-LITERAL clause and suffixed with S unless the literal ends in CH, S, SH, or X, in which case it is suffixed with ES. For example, BATCH would be extended to BATCHES. If the standard literal is more than 30 or 31 characters long (dependent on its last character) the default is rejected and you cannot construct the RIM.

If the interrogate keyword defined in a first level MEMBER-TYPE is used in an interrogation then the plural literal of that MEMBER-TYPE is displayed in both the output headers and footers.

If the interrogate keyword defined in a second level MEMBER-TYPE is used in an interrogation then the plural literal defined in the first level MEMBER-TYPE is displayed in the header and the plural literal defined in the second level MEMBER-TYPE is displayed in the footer.

Refer to "Defining a Base or User-defined Member Type" on page 286 for details of the two level member-types.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of the GLOSSARY and LIST commands.

Specifying Generic User-defined Attributes

To define the user-defined attributes that can be included in the definition of both a base member type and the user-defined member types based on it, specify:

`GENERIC-ATTRIBUTES` *`attribute-list`*

where *`attribute-list`* is:

- The names of one or more ATTRIBUTE-TYPE or ATTRIBUTE-GROUP members. All attributes defined by the named ATTRIBUTE-TYPEs or by the ATTRIBUTE-TYPEs directly and indirectly contained in the named ATTRIBUTE-GROUPs can be included in the definition of the member types. Each member name must be separated by a comma if entered via the command interface.
- One or more sets of mutually exclusive ATTRIBUTE-TYPE members. Each set must contain two or more ATTRIBUTE-TYPE member names each separated by an ELSE keyword. The same ATTRIBUTE-TYPE member can be specified in more than one set. Only one of the attributes defined by the alternative ATTRIBUTE-TYPE members can be present in the definition of the member types. Each set must be separated by a comma if entered via the command interface.

The GENERIC-ATTRIBUTES clause can only be specified in an ASG-supplied base MEMBER-TYPE (containing an IS clause).

Each individual ATTRIBUTE-TYPE or ATTRIBUTE-GROUP member name or each set of alternative ATTRIBUTE-TYPE members can be followed with:

- OPTIONAL NO to specify that an attribute or all the attributes contained in the named group, or one of the mutually exclusive attributes within a set must be present when a member is encoded
- OPTIONAL YES to specify that the attribute(s) need not be present
- OPTIONAL WARN to specify that the attribute(s) need not be present but, if not a warning message displays

The default is OPTIONAL YES.

The GENERIC-ATTRIBUTES clause is optional.

Specifying Non-Generic User-defined Attributes

To define the user-defined attributes that can be included in the definition of the member type, specify:

`ATTRIBUTES` *`attribute-list`*

where *`attribute-list`* is:

- The names of one or more ATTRIBUTE-TYPE or ATTRIBUTE-GROUP members. All attributes defined by the named ATTRIBUTE-TYPEs or by the ATTRIBUTE-TYPEs directly and indirectly contained in the named ATTRIBUTE-GROUPs can be included in the definition of the member type. Separate each member name with a comma if entering via the command interface.
- One or more sets of mutually exclusive ATTRIBUTE-TYPE members. Each contains two or more ATTRIBUTE-TYPE member names each separated by an ELSE keyword. You can specify the same ATTRIBUTE-TYPE member name in more than one set. Only one of the attributes defined by the alternative ATTRIBUTE-TYPE members can be present in the definition of the member type. Separate each member name with a comma if entering via the command interface..

Each individual ATTRIBUTE-TYPE or ATTRIBUTE-GROUP member name or each set of alternative ATTRIBUTE-TYPE members can be followed with:

- OPTIONAL NO to specify that an attribute or all the attributes contained in the named group, or one of the mutually exclusive attributes within a set, must be present when a member is encoded
- OPTIONAL YES to specify that the attribute(s) need not be present
- OPTIONAL WARN to specify that the attribute(s) need not be present but, if not a warning message displays.

The default is OPTIONAL YES and the ATTRIBUTES clause is optional.

Defining a Member Type Level Number

To define a member type level number, specify: `LEVEL n`

where *n* is an unsigned integer in the range 0 to 255.

The level number defines the hierarchical position of a base member type and the user-defined member types based on it.

ASG recommends that level numbers are defined in multiples of 5 or 10 in order to leave slots into which other member types can be inserted at a later date.

With the exception of the SEE, UDR, and UDRS clauses a member cannot refer via any clause in its definition to another member based on the same member type if the referenced member's member type has a lower level number. A member can refer via its SEE, UDR, and UDRS clauses to any other member regardless of that members member type level number.

A member can only refer to another member based on the same base member type that has the same level number if that number is 0.

The LEVEL clause does not restrict references between different base member types or user-defined member types based on the different base member types.

You can use the RELATIONSHIPS-VIA clause to override the effect of the LEVEL clause for any clause of a member type.

The LEVEL clause is optional.

Disallowing Relationships Between Members of the Same Member Type

To allow or disallow relationships between members of the same member type, specify:

`RECURSIVE option`

where *option* is YES or NO.

Specify YES to allow this type of relationship. Specify NO to disallow this type of relationship. YES is the default.

The RECURSIVE clause does not restrict relationships via the SEE, UDR, and UDRS clauses.

The RELATIONSHIPS-VIA clause overrides the effect of the RECURSIVE clause for any clause of a member type.

The RECURSIVE clause is optional.

Allowing and Disallowing Relationships Via Specified Clauses

By default a member type can contain any of the clauses available in the base member type which it is equivalent to or based upon. By default these clauses can refer to:

- Any of the member types to which the clause can refer in the base member type
- Any user-defined member types based on those member types

To disallow references from a member type via a particular clause, specify:

```
RELATIONSHIPS VIA clause DISALLOWED
```

where *clause* is any clause allowed in the base member type that defines a relationship between members.

To restrict the member types to which a reference can be made via a particular clause, specify:

```
RELATIONSHIPS VIA clause ALLOW member-name-list
```

where *member-name-list* is a list of one or more MEMBER-TYPE, MEMBER-TYPE-GROUP, RELATIONSHIP-TYPE, or RELATIONSHIP-GROUP member names. Each member name must be separated by a comma if entered via the command interface.

Only the member types defined by the named MEMBER-TYPEs and RELATIONSHIP-TYPEs or contained by the named MEMBER-TYPE-GROUPs and RELATIONSHIP-GROUPs, can be referenced via clause.

If a member refers to a member that does not have an encoded source record, then a dummy member is created. The member type of the dummy member is determined by the clause and the type of member in which it specified.

If you specify an ALLOW clause then the member type of dummy members is defined by the first member named in the member-name-list. If the first member is a MEMBER-TYPE-GROUP or RELATIONSHIP-GROUP, then the dummy member type is defined by the first MEMBER-TYPE or RELATIONSHIP-TYPE they contain.

To explicitly define the type of dummy member created by a reference via a particular clause, specify:

```
RELATIONSHIPS VIA clause DUMMY member-name
```

or

```
RELATIONSHIPS VIA clause ALLOW member-name-list
DUMMY member-name
```

where *member-name* is a MEMBER-TYPE or RELATIONSHIP-TYPE member name.

You can specify several RELATIONSHIP VIA clauses in a MEMBER-TYPE definition each defining the type of relationships permitted via a particular clause.

The RELATIONSHIPS-VIA clause is optional.

Automatically Defining EA Relationships

To automatically define EA relationships from a member type, specify:

```
AUTO-REF-STRING string
```

where *string*:

- Is a string of up to 80 characters
- Defines an indicator which if placed one or more character spaces before a member name in an updated member's definition, automatically creates an EA relationship between the updated and prefixed member
- Must be delimited if entered via the command interface

The EA relationship is via whichever clause specified in the MEMBER-TYPE's SEE clause is defined in an ATTRIBUTE-TYPE member having the following clauses:

```
EDIT-CODE-1 6
EDIT-CODE-2 6
SKELETON-CODE 8
```

For example, if you define a SEE string for use in ITEM members as follows:

```
AUTO-REF-STRING SEE
```

and define that references are to be via the SEE clause:

```
ATTRIBUTE-TYPE
FREE-FORM-TEXT
IDENTIFIED-BY SEE
EDIT-CODE-1 6
EDIT-CODE-2 6
SKELETON-CODE 8
SKELETON-HELP AUTOMATICALLY MAINTAINED EA RELATIONSHIPS
```

and then specify SEE in the NOTE clause of an ITEM member named IT-ONE:

```
NOTE
SEE IT-TWO AND SEE IT-THREE
```

then the clause:

```
SEE
IT-TWO
IT-THREE
```

is automatically generated in the definition of IT-ONE.

Only one AUTO-REF-STRING clause can be specified in a MEMBER-TYPE definition. Only one ATTRIBUTE-TYPE member in that MEMBER-TYPE's SEE clause can be defined for use by the AUTO-REF-STRING clause.

Any number of members can be referenced in this manner. Because these relationships are automatically maintained, the referenced member's names are not listed under the relationship clause in the assisted update buffer.

The AUTO-REF-STRING clause is optional.

Preventing a Member Type Being Displayed in the Panel Interface/Displaying IMS Collective Member Types

To prevent a member type being displayed in the panel interface, specify:

```
LONG-NAME MMR-DISAPPEAR
```

MMR-DISAPPEAR must be delimited if entered via the command interface.

The member type is included in your RIM but, because it is not displayed in selection list panels, it is not visible to users of the panel interface.

You therefore have the ability to continue to include obsolescent member types in your RIM.

You can still define an ENCODE-KEYWORDS clause for the member type and so add members to the repository via the command interface using commands such as UPDATE and RESTORE.

Some ADABAS, IMS, MARKIV, and TOTAL member types are defined in two levels of base MEMBER-TYPE. You must specify a LONG-NAME MMR-DISAPPEAR clause in the definition of the first level member type so as to prevent its being displayed in the panel interface.

To display an IMS collective member type in the panel interface, specify:

```
LONG-NAME encode-keyword
```

where *encode-keyword*:

- Must be identical to the encode keyword of the second level IMS member type for which it is specified. The member type must also be specified in a HIERARCHY COLLECTIVE clause.
- Must be delimited if entered via the command interface.

IMS collective member types are identified with the character C in selection list panels.

The LONG-NAME clause is optional but in order to enable ToolSet Services each MEMBER-TYPE must have either an ENCODE-KEYWORDS or a LONG-NAME clause defined.

Refer to "Defining a Base or User-defined Member Type" on page 286 for details of the two level MEMBER-TYPEs.

Defining Naming Conventions for Entity Members

To define naming conventions for entity members, specify:

NAMING *convention-strings*

where *convention-strings*:

- Are one or more strings each containing:
  - Characters from the standard and/or extended character sets for names
  - Seven optional special characters, three of which are wildcards (* _ #),three of which restrict the length of the member name (> < =), and one of which allows you to incorporate variables into names (:)
- Must each be delimited if entered via the command interface.

The naming conventions restrict the range of names that can be used for a member type. If, during an assisted update for a particular member type, you enter a member name that does not conform to the convention, the name is disallowed.

The naming conventions are displayed on selection list panels and enforced for updates made using both the AUPD line command and the AUPDATE primary command.

Repository member names can contain up to 32 characters from the standard or extended character sets for names. You must not define a convention which results in users attempting to create members with invalid names. Names containing only numeric characters or any character from the extended character set beginning with an underscore or hyphen, must be delimited if specified in command interface functions.

The three wildcard characters you can use to specify what can appear at the corresponding positions in the member name, are as follows:

- A wildcard for open naming conventions: the position must be filled with one character. The default character is an underscore (_).
- An optional wildcard: the position can be filled with any number of characters, or none at all. The default character is an asterisk (*).
- A wildcard for numeric characters: the position must be filled with a numeric character. The default character is a hash (#).

For example, specifying the following in the definition of the MEMBER-TYPE member defining the DATAFLOW member type:

```
NAMING DF_#-*
```

would mean that the names of all DATAFLOW members must begin with DF, immediately followed by any character, then a number and then a hyphen, and ending with up to 27 alphanumeric characters, since the total maximum is 32 characters.

You can also limit the length of member names by including any of three special characters in the NAMING clause. The characters are as follows:

- A maximum name delimiter, whose position determines the maximum allowed length of a name: the name must end before the position occupied by this character. The default character is the less than symbol (<).
- An exact length delimiter, which restricts names to a particular length: the last character of the name must immediately precede this character. The default character is the equals sign (=).
- A minimum name delimiter: the end of the name must come after the position occupied by this character. The default is the greater than symbol (>).

The less than symbol (<) and the greater than symbol (>) cannot be combined in a single naming convention.

The following are examples of how the length of member names can be restricted.

To define that a name must begin with MEM- followed by exactly 3 characters, specify:

```
NAMING MEM-___=
```

If the equals sign (=) were not used, the name could be of any length.

To define that a name must begin with USER followed by no more than two numeric characters, specify:

```
NAMING USER##<
```

To define that a name can comprise any characters, but must be between 2 and 4 characters long, specify:

```
NAMING __**<
```

You can also include variables in the NAMING clause; at run-time, the values of the variables are included in the displayed naming convention. This allows for dynamic naming conventions, which change during a session or according to the user or date. For example, you could include a variable that contains the user's ID, so that users can only access members created by themselves.

The variable name must be enclosed by the wildcard character for the variable part of naming conventions, which defaults to a colon (:). For example, with the variable &USER set to CON, the following clause:

```
NAMING IT-:&USER:##<
```

would be, in effect:

```
NAMING IT-CON##<
```

If you want only part of the value of a variable to be included, you can specify which part, in parentheses, after the variable name:

```
NAMING :variable(first-char,length):
```

For example:

```
NAMING IT-:&USER(2,2):##<
```

would become:

```
NAMING IT-ON##<
```

with only the second and third characters of the variable &USER being included.

You can specify for a particular naming convention to be used only if a particular condition is met, by using:

NAMING IF *condition* THEN *convention-string*

where:

condition is a Boolean expression.

convention-string is a naming convention string, as defined above.

For example:

```
NAMING IF &USER = MASTER THEN CO_____IMS
```

would result in the naming convention CO_____IMS only being used if the variable USER had the value MASTER. Otherwise, if this was the only NAMING clause in the MEMBER-TYPE definition, then the member type would not appear in the selection list.

Any number of conditional NAMING clauses can be included in a MEMBER-TYPE definition to cover every eventuality.

The seven special characters are set in the following variables in SEXEC EC1060:

- the > character is set in variable MDG_NAMOL
- the < character is set in variable MDG_NAMSOL
- the = character is set in variable MDG_NAMON
- the * character is set in variable MDG_NAMOPT
- the _ character is set in variable MDG_NAMJOK
- the # character is set in variable MDG_NAMNUM
- the: character is set in variable MDG_NAMVAR

You can tailor the special characters using the environment functions (provided by panel TZ42300).

The NAMING clause is optional.

Defining Complex Naming Conventions

To take a user exit for complex naming conventions, specify:

```
NAMING-EXIT member-name
```

where *member-name*:

- Is the name of an EXECUTIVE member on the MP-AID
- Must be delimited if entered via the command interface

The user exit routine is defined in the SEXEC member from which the EXECUTIVE member is constructed.

The NAMING-EXIT clause is optional. If it is, specified it will override any naming conventions specified in the NAMING clause.

Refer to Chapter 7, "User Exits," on page 171 for details of how to define a user exit routine to check names.

Specifying the Clauses and Keywords Displayed During Assisted Update

To define the clauses and keywords displayed during an assisted update of the member type, specify:

```
SEE attribute-list
```

where *attribute-list* is a list of one or more ATTRIBUTE-TYPE and ATTRIBUTE-GROUP member names. Each member name must be separated by a comma if entered via the command interface.

Clauses and keywords are displayed in the assisted update buffer and documented in the in-context help in the order that the ATTRIBUTE-TYPE members defining them are listed in the SEE clause (whether directly or indirectly via an ATTRIBUTE-GROUP).

If an ATTRIBUTE-GROUP member is specified then the clauses and keywords are displayed in the order that the ATTRIBUTE-TYPE members defining them are directly or indirectly contained in the group.

All clauses and keywords which you want to update via an assisted update buffer must be specified in the SEE clause.

If you have renamed UDR or UDRS clauses then the ATTRIBUTE-TYPE members in which they are defined must also be specified in the SEE clause of the HIERARCHY member containing the MEMBER-TYPE.

An ATTRIBUTE-TYPE member renaming a UDRS subclause must not be specified in a MEMBER-TYPE SEE clause. In order to display a UDRS subclause in an assisted update buffer you must specify it in the SKELETON-TEXT clause of the ATTRIBUTE-TYPE members defining the UDR clauses.

A clause that can contain free form text as its value must be specified last in the attribute-list.

The SEE clause is optional but assisted update buffers and in-context help are only available for member types for which it is specified.

Defining an Alias Identifier

To define an alias identifier for the member type, specify:

```
ALIAS alias
```

where *alias*:

- Is a two character string and must be unique in the Administration Repository
- Must be delimited if entered via the command interface

The alias provides a short identifier that can be used to reference the member type from within macros and user exit routines.

The ALIAS clause is mandatory. You cannot enable ToolSet Services if the ALIAS clause is not specified.

Documenting Help for a Member Type

To document help for a member type, specify:

```
HELP text
```

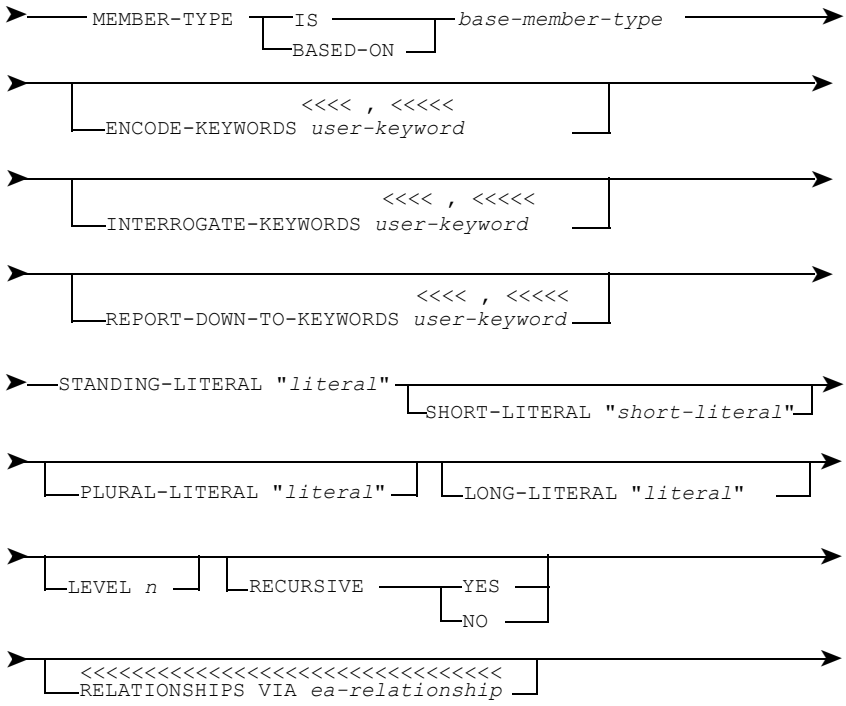
where *text*:

- Is free form text documenting the member type
- Is displayed as in-context help from within an assisted update buffer or in response to an MTHELP command

Help on the clauses and keywords within the member type must be defined in the HELP clauses of the relevant ATTRIBUTE-TYPE members.

The HELP clause must be the last clause specified in a MEMBER-TYPE definition.

MEMBER-TYPE Syntax



```
➤──┬──────────────
```

mem-type is a MEMBER-TYPE member name.

mem-type-gp is a MEMBER-TYPE-GROUP member name.

rel-type is a RELATIONSHIP-TYPE member name.

rel-grp is a RELATIONSHIP-GROUP member name.

attributes-clause is:

```
➤──┬─ attr-group ──────────────────────┬──┬──────────────────────────────┬──➤
   └─ attr-type ──┬──────────────────┬──┘  ├─ OPTIONAL YES ─────────────┤
                  ├<<<<<<<<<<<<<<<<<<┤     ├─ OPTIONAL NO ──────────────┤
                  └─ ELSE attr-type ─┘     └─ OPTIONAL WARN ────────────┘
```

where:

attr-group is an ATTRIBUTE-GROUP member name.

attr-type is an ATTRIBUTE-TYPE member name.

ref-string is a string of up to 80 characters.

encode-keyword is the encode keyword of a second level IMS member type.

convention is a string of characters from the standard and extended character set for names, optionally including the special characters # * _ > < = :.

executive is an EXECUTIVE member name.

attr-type and *attr-group* are as defined above.

common-clauses are any of the clauses common to all member types except for the SEE and ALIAS clauses which are used for special purposes in the MEMBER-TYPE member type.

help is one or more lines of free form text.

Note:

The commas and delimiters shown in the above syntax are required when creating MEMBER-TYPE members via the command interface.

Refer to *ASG-ControlManager User's Guide* for details of the character set for names.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of the common clauses.

MEMBER-TYPE-GROUP

MEMBER-TYPE-GROUP defines a logical group of member types.

Refer to "MEMBER-TYPE-GROUP Syntax" on page 309 for the syntax of the MEMBER-TYPE-GROUP member type.

By defining a MEMBER-TYPE-GROUP you can reference member types collectively and so simplify your repository information model (RIM) definition.

Each MEMBER-TYPE-GROUP named in the CONTAINS clause of the HIERARCHY member defining the RIM defines an option on the member type cluster menu. The member types specified in the SEE clause of the MEMBER-TYPE-GROUP are listed on the selection list panel selected from the option on the cluster menu. Members having any of the listed member types can, for example, be listed, printed, removed, created, or updated via the selection list panel.

You therefore have the ability to define a logical group of related member types and at the same time provide an interface that enables users to process members having any of the grouped member types.

Specifying the Entity Member Types Contained in the Group

To define the entity member types contained in the group, specify:

`CONTAINS` *`member-type-list`*

where *`member-type-list`* is one or more MEMBER-TYPE and MEMBER-TYPE-GROUP member names. Each member name must be separated by a comma if entered via the command interface.

If a MEMBER-TYPE-GROUP is specified then all the MEMBER-TYPEs it directly and indirectly contains are included in the group.

The same MEMBER-TYPE can be contained in different MEMBER-TYPE-GROUPs.

The CONTAINS clause is mandatory.

Defining a Member Type Cluster Menu Option

To define an option on the cluster menu, specify:

```
OPTION option
```

where *option*:

- Is either a string of up to two characters or one or more IF directives each in the format IF condition THEN option and specified on a separate line
- Must be delimited if entered via the command interface

ASG recommends defining numeric options in the range 1 to 9. If other characters are defined then you must also enter them after the parameters CLUSTER INDEX passed to MTLIST in the CALLS clause of the following FMT-SCREEN members: SC-TD10000, SC-TD50000, SC-TK10000, SC-TK50000, SC-TW10000, and SC-TW50000.

The following alphabetic options are reserved for use by ASG-supplied MEMBER-TYPE-GROUP members: DC, LC, PJ, and WB.

You must ensure that different MEMBER-TYPE-GROUP members defining a single cluster menu do not share the same option.

You can use the IF directive to specify the conditions under which the option is displayed. For example, suppose you had three MEMBER-TYPE-GROUPs named MTG1, MTG2, and MTG3 and you wanted to restrict the option defined by MTG2 to users with the master password. You could achieve this by defining:

```
OPTION   1 in MTG1

OPTION   IF &USER EQ MASTER THEN 2 in MTG2

OPTION   IF &USER EQ MASTER THEN 3

         IF &USER NE MASTER THEN 2 in MTG3
```

Refer to *ASG-Manager Products Procedures Language* for details of the IF directive.

To define an option name on the member type cluster menu, specify:

```
OPTION-NAME name
```

where *name*:

- Is a string of up to 15 characters
- Must be delimited if entered via the command interface

To define an option description on the member type cluster menu, specify:

```
OPTION-TEXT description
```

where *description*:

- Is a string of up to 50 characters
- Must be delimited if entered via the command interface

name and *description* will also form the title of the member type selection list panel selected from the option on the cluster menu.

The OPTION, OPTION-NAME, and OPTION-TEXT clauses are optional. If they are omitted the grouped member types will be included in the RIM but will not appear as an option on the member type cluster menu.

Specifying the Member Types Selected from the Cluster Menu

To define the member types listed on the selection list panel selected from the cluster menu, specify:

```
SEE member-type-list
```

where *member-type-list* is one or more MEMBER-TYPE and RELATIONSHIP-TYPE member names. Each member name must be separated by a comma if entered via the command interface.

The MEMBER-TYPE members define the entity member types listed on the selection list panel. The RELATIONSHIP-TYPE members define the ER relationship types permitted between the listed entity member types.

Any RELATIONSHIP-TYPE members directly and indirectly specified in the RELATIONSHIPS and ALTERNATIVE-RELATIONSHIPS clauses of the HIERARCHY and RELATIONSHIP-GROUP members defining the RIM that are not also specified in the SEE clause of a MEMBER-TYPE-GROUP member are selected from option:

```
N None   Types Without Cluster
```

on the member type cluster menu.

The SEE clause is optional. If it is omitted you will not be able to process members via the selection list panel selected from the cluster menu.

MEMBER-TYPE-GROUP Syntax

```
➤──MEMBER-TYPE-GROUP──────────────────────────────────────────────────➤
                   <<<<<, <<<<<<<
➤── CONTAINS ──┬─mem-type-group─┬──┬──────────────────────────────────➤
                └─mem-type───────┘  │              <<< , <<<
                                    └──SEE ──┬─mem-type─┬──┘
                                             └─rel-type─┘
➤──┬──────────────────────────┬──┬──────────────────────────────────➤
   └─OPTION "option"──────────┘  └─OPTION-NAME "name"──────────────┘
➤──┬──────────────────────────────┬──┬──────────────────────────────➤
   └─OPTION-TEXT "description"────┘  └─common-clauses──────────────┘
➤──┬─;─┬──────────────────────────────────────────────────────────────➤
   └─.─┘
```

where:

mem-type-group is a MEMBER-TYPE-GROUP member name.

mem-type is a MEMBER-TYPE member name.

rel-type is a RELATIONSHIP-TYPE member name.

option is a string of up to 2 characters.

name is a string of up to 15 characters.

description is a string of up to 50 characters.

common-clauses are any of the clauses available in all member types with the exception of the SEE clause which is used in the MEMBER-TYPE-GROUP member type for special purposes

Note:

The commas and delimiters shown in the above syntax are required when defining MEMBER-TYPE-GROUP members via the command interface.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of the common clauses.

RELATIONSHIP-CLASS

RELATIONSHIP-CLASS defines a relationship type class.

To define the name of the relationship type class, specify:

```
PRIMARY-NAME name
INVERSE-NAME name
```

where *name*:

- Can contain up to 32 characters from the standard character set for names but must not begin with an underscore
- Cannot be the same as the PRIMARY-NAME or INVERSE-NAME of any other RELATIONSHIP-TYPE or RELATIONSHIP-CLASS member defining the RIM

All RELATIONSHIP-TYPE members that refer via their CLASS clause to a RELATIONSHIP-CLASS member belong to the class that it defines.

You can interrogate the relationship members belonging to the relationship type class by specifying the PRIMARY-NAME in WHICH VIA and DRETRIEVE VIA commands and any panel interface functions that internally execute these commands. The INVERSE-NAME can be specified in the WHICH VIA command. The names should therefore be meaningful.

The PRIMARY-NAME clause is mandatory. The INVERSE-NAME clause is optional.

Refer to *ASG-ControlManager User's Guide* for details of the standard character set.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of the WHICH command.

Refer to *ASG-Manager Products Procedures Language* for details of the DRETRIEVE command.

RELATIONSHIP-CLASS Syntax

```
>>---RELATIONSHIP-CLASS ------------------------------------------>
>---PRIMARY-NAME name ---+-------------------------+---------------->
                         |  INVERSE-NAME name      |
>---+-------------------------------------------------------------->
    |  common clause                                |
>---+---+---;---+-------------------------------------------------->
        |   .   |
```

where:

name can contain up to 32 characters from the standard character set for names but must not begin with an underscore.

common-clauses are any of the clauses common to all member types.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of the common clauses.

Refer to *ASG-ControlManager User's Guide* for details of the character set for names.

RELATIONSHIP-GROUP

RELATIONSHIP-GROUP defines a group of relationship member types.

Refer to "RELATIONSHIP-GROUP Syntax" on page 313 for the syntax of the RELATIONSHIP-GROUP member type.

The grouped RELATIONSHIP-TYPE members can be referenced collectively thus simplifying your RIM definition. For example, HIERARCHY members can refer to RELATIONSHIP-GROUP members via the RELATIONSHIPS clause.

Defining a Group of Relationship Member Types

To define a group of relationship member types, specify:

```
RELATIONSHIPS relationship-type-list
```

relationship-type-list is a list of one or more RELATIONSHIP-TYPE or RELATIONSHIP-GROUP member names. Each member name must be separated by a comma if entered via the command interface.

All relationship member types defined by the specified RELATIONSHIP-TYPEs, or by the RELATIONSHIP-TYPEs directly and indirectly contained in the specified RELATIONSHIP-GROUPs, are included in the group. A RELATIONSHIP-TYPE can be contained by more than one RELATIONSHIP-GROUP.

All the RELATIONSHIP-TYPE members directly and indirectly specified in the RELATIONSHIPS clause should also be specified in the SEE clause of the MEMBER-TYPE-GROUP member that groups their source and target member types.

If a RELATIONSHIP-TYPE member is specified in an ALTERNATIVE-RELATIONSHIPS clause then it need not be specified in a RELATIONSHIPS clause.

The RELATIONSHIPS clause is optional.

Defining Mutually Exclusive Relationship Member Types

To define a set of mutually exclusive relationship member types, specify:

```
ALTERNATIVE-RELATIONSHIPS relationship-type-list
```

where *relationship-type-list* is two or more RELATIONSHIP-TYPE member names each separated by an ELSE keyword.

For example, if you specified:

```
ALTERNATIVE-RELATIONSHIPS SYSTEM-INPUTS-FILE ELSE
PROGRAM-OUTPUTS-FILE ELSE MODULE-UPDATES-FILE
```

then the same FILE member could not be the target of two relationship members having any two of the three alternative relationship member types.

Multiple sets of mutually exclusive relationship member types can be defined. The same RELATIONSHIP-TYPE member can be specified in more than one set. Each set must be separated by a comma if entered via the command interface.

Mutually exclusive RELATIONSHIP-TYPE members must share at least one common MEMBER-TYPE or RELATIONSHIP-TYPE member in their SOURCE or TARGET clause.

If a RELATIONSHIP-TYPE member has the same MEMBER-TYPE or RELATIONSHIP-TYPE as both its source and target, then you must specify the SOURCE or TARGET keywords in the ALTERNATIVE-RELATIONSHIPS clause to indicate whether it is the source or the target that is Mutually exclusive.

For example, to define that the target of PROGRAM-GENERATES-PROGRAM is mutually exclusive with the source of PROGRAM-GENERATES-REPORT, specify:

```
ALTERNATIVE-RELATIONSHIPS PROGRAM-GENERATES-PROGRAM
TARGET ELSE PROGRAM-GENERATES-REPORT
```

A PROGRAM member could not be both the target of a PROGRAM-GENERATES-PROGRAM member and the source of a PROGRAM-GENERATES-REPORT member but could be the source of both a PROGRAM-GENERATES-PROGRAM and a PROGRAM-GENERATES-REPORT member.

To define that the source of both PROGRAM-GENERATES-PROGRAM and PROGRAM-GENERATES-REPORT are mutually exclusive, specify:

```
ALTERNATIVE-RELATIONSHIPS PROGRAM-GENERATES-PROGRAM
SOURCE ELSE PROGRAM-GENERATES-REPORT
```

To define that both the source and target of PROGRAM-GENERATES-PROGRAM is mutually exclusive with the source of PROGRAM-GENERATES-REPORT, specify:

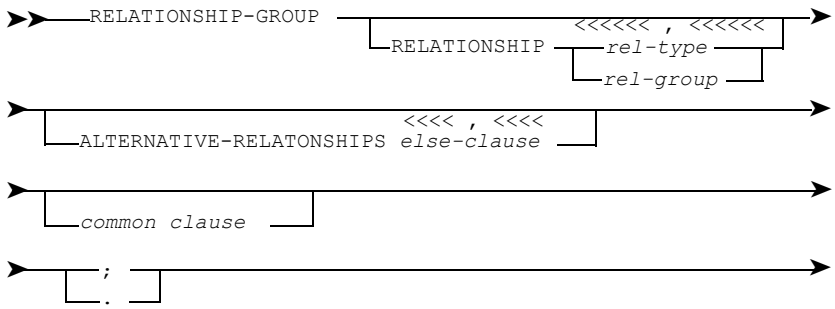
```
ALTERNATIVE-RELATIONSHIPS PROGRAM-GENERATES-PROGRAM
SOURCE ELSE PROGRAM-GENERATES-PROGRAM TARGET ELSE
PROGRAM-GENERATES-REPORT
```

All the RELATIONSHIP-TYPE members specified in the ALTERNATIVE-RELATIONSHIPS clause should also be specified in the SEE clause of the MEMBER-TYPE-GROUP member that groups their source and target member types.

If a RELATIONSHIP-TYPE member is specified in an ALTERNATIVE-RELATIONSHIPS clause then it need not be specified in a RELATIONSHIPS clause.

The ALTERNATIVE-RELATIONSHIPS clause is optional.

RELATIONSHIP-GROUP Syntax



where:

rel-type is a RELATIONSHIP-TYPE member name.

rel-group is a RELATIONSHIP-GROUP member name.

else-clause is:

```
                                  <<<<<<<<<<<<<<<<<<<<<<<<<<<<
 ►──rel-type ──┬──────────┬──────────┬──ELSE rel-type ──┬──────────┬──────────┬──►
               │ SOURCE   │          │                  │ SOURCE   │          │
               │ TARGET   │          │                  │ TARGET   │          │
```

where *rel-type* is as defined above.

common-clauses are any of the clauses common to all member types.

Note:

The commas shown in the above syntax are required when defining RELATIONSHIP-GROUP members via the command interface.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of the common clauses.

RELATIONSHIP-TYPE

RELATIONSHIP-TYPE defines a relationship member type.

Refer to "RELATIONSHIP-TYPE Syntax" on page 332 for the syntax of the RELATIONSHIP-TYPE member type.

The RELATIONSHIP-TYPE member type defines a relationship member type in your repository information model (RIM) that represents a particular ER relationship type.

Up to 12192 relationship member types can be defined in a single RIM. An unlimited number of members having the defined member type can be entered in a repository to which the RIM is applied.

By default, you can interrogate all relationship members of any member type by specifying the interrogate keyword RELATIONSHIPS in any function that can be applied to a selection of members according to their member type. For example, to list all relationship members enter LIST RELATIONSHIPS. The keyword RELATIONSHIPS is also displayed in the headings and totals output of the GLOSSARY and LIST commands (and any panel interface functions that internally execute these commands).

You can define alternative interrogate keywords and user-defined attributes that are common to all relationship member types, by including the following MEMBER-TYPE in your RIM:

```
MEMBER-TYPE
IS RELATIONSHIP
INTERROGATE-KEYWORD keyword-list
STANDARD-LITERAL literal
PLURAL-LITERAL literal
GENERIC-ATTRIBUTES attribute-list
ALIAS alias
LONG-NAME MMR-DISAPPEAR
```

For example, if you specified:

```
INTERROGATE-KEYWORD ER-RELATIONSHIPS
```

then the keyword ER-RELATIONSHIPS would replace RELATIONSHIPS.

Refer to the MEMBER-TYPE member definition for full details of the above clauses.

Naming the Relationship Member Type

To define the name of the relationship member type, specify:

```
PRIMARY-NAME name
INVERSE-NAME name
```

where *name* can contain up to 32 characters from the standard character set for names but must not begin with an underscore.

If you do not also define a PLURAL-LITERAL clause then name can be:

- 30 characters long if the last character of the name is CH, S, SH, or X.
- 31 characters long if the last character of the name is not CH, S, SH, or X.

PRIMARY-NAME cannot be the same as:

- The PRIMARY-NAME or INVERSE-NAME of any other RELATIONSHIP-TYPE or RELATIONSHIP-CLASS member defining the RIM.
- The ENCODE-KEYWORDS or INTERROGATE-KEYWORDS of any MEMBER-TYPE member.

INVERSE-NAME cannot be the same as the PRIMARY-NAME or INVERSE-NAME of any RELATIONSHIP-TYPE or RELATIONSHIP-CLASS member defining the RIM.

The definition of relationship members must begin with the PRIMARY-NAME.

The PRIMARY-NAME can be specified in all functions that can be applied to a selection of members according to their member type. For example, you could list relationship members having a relationship member type of MODULE-UPDATES-FILE by entering:

```
LIST MODULE-UPDATES-FILE;
```

You can also interrogate relationships via a particular relationship type by specifying the PRIMARY-NAME or INVERSE-NAME in a WHICH VIA or DRETRIEVE VIA command, or in any panel interface functions that internally execute these commands.

The PRIMARY-NAME is displayed in the output of interrogations of the relationship member type unless you have defined an alternative name in the PLURAL-LITERAL, SHORT-LITERAL, or STANDARD-LITERAL clauses.

The PRIMARY-NAME clause is mandatory. The INVERSE-NAME clause is optional.

Refer to *ASG-ControlManager User's Guide* for details of the standard character set.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of the WHICH and LIST commands.

Refer to *ASG-Manager Products Procedures Language* for details of the DRETRIEVE command.

Defining the Relationship Type Class

To define the relationship type class to which the relationship member type belongs, specify:

```
CLASS relationship-class
```

where *`relationship-class`* is the name of a RELATIONSHIP-CLASS member.

The PRIMARY-NAME and INVERSE-NAME of the RELATIONSHIP-CLASS member can be interrogated by the WHICH VIA or DRETRIEVE VIA commands and any panel interface functions that internally execute these commands.

The CLASS clause is optional.

Tailoring GLOSSARY, REPORT, WHAT, and WHICH Output

To define a name that identifies the relationship member type in the output from the GLOSSARY, REPORT, WHAT and WHICH commands (and any panel interface functions that internally execute these commands), specify:

`STANDARD-LITERAL literal`

where *literal*:

- Can be a maximum of 32 characters long. If you do not also define a PLURAL-LITERAL clause then literal can be either:
  - 30 characters long if the last character of the literal is CH,S. SH, or X
  - 1 characters long if the last character of the literal is not CH, S. SH, or X.
- Must be delimited if entered via the command interface.

The STANDARD-LITERAL clause is optional. The PRIMARY-NAME displays if you do not specify a STANDARD-LITERAL clause.

Tailoring LIST Output

To define a name that identifies the relationship member type in the TYPE column of output from the LIST command (and any panel interface functions that internally execute this command), specify:

`SHORT-LITERAL literal`

where *literal*:

- Can be a maximum of 12 characters long
- Must be delimited if entered via the command interface

The SHORT-LITERAL clause is optional. If you do not specify a SHORT-LITERAL clause then the default is taken from either:

- The STANDARD-LITERAL clause
- The PRIMARY-NAME clause if no STANDARD-LITERAL clause is specified

and, if necessary, abbreviated to 12 characters.

Tailoring LIST and GLOSSARY Headings Output

To define a name that identifies the relationship member type in the headings output of the LIST and GLOSSARY commands (and any panel interface functions that internally execute these commands), specify:

```
PLURAL-LITERAL literal
```

where *literal*:

- Can be a maximum of 32 characters long
- Must be delimited if entered via the command interface

The PLURAL-LITERAL clause is optional. If you do not specify a PLURAL-LITERAL clause then the default is taken from either:

- The STANDARD-LITERAL clause
- The PRIMARY-NAME clause if no STANDARD-LITERAL clause is specified

and suffixed with S unless the defaults taken end in CH, S, SH or X, in which case it is suffixed with ES. For example, BATCH would be extended to BATCHES. If the resulting name is greater than 32 characters then the RIM cannot be constructed onto the MP-AID.

Defining the Source and Target Member Types

To define the member types that can be the source of the relationship member type, specify:

```
SOURCE TYPE member-type-list
```

To define the member types that can be the target of the relationship member type, specify:

```
TARGET TYPE member-type-list
```

where *member-type-list* can be one or more MEMBER-TYPE, MEMBER-TYPE-GROUP, RELATIONSHIP-TYPE, or RELATIONSHIP-TYPE-GROUP member names. Each member name must be separated by a comma if entered via the command interface.

If a MEMBER-TYPE-GROUP or a RELATIONSHIP-GROUP member is specified then all the member types defined by the MEMBER-TYPE or RELATIONSHIP-TYPE members they directly or indirectly contain can be the source or target.

If a relationship member's source or target does not have an encoded source record, then a dummy member is created. The member type of the dummy is defined by the first member specified in the member-type-list. If the first member is a MEMBER-TYPE-GROUP or RELATIONSHIP-GROUP, then the dummy member type is defined by the first MEMBER-TYPE or RELATIONSHIP-TYPE member they contain.

If you want dummy members to have a particular member type you should ensure that the required member type is the first one listed.

The SOURCE and TARGET clauses are mandatory.

Disallowing Unencoded Source and Target Members

To prevent members without an encoded source record being validated as the source or target of the relationship member type, specify:

```
SOURCE TYPE member-type-list NO-DUMMY
TARGET TYPE member-type-list NO-DUMMY
```

Members used by the source or target member are not required to have an encoded source record.

The NO-DUMMY clause is optional.

Defining a Permitted Number of Relationships

To define the number of times that source and target members can refer to one another via the relationship member type, specify:

```
SOURCE TYPE member-type-list CARDINALITY min TO max
TARGET TYPE member-type-list CARDINALITY min TO max
```

where *min* and *max* can be either MANY or any integer. MANY is the default and specifies that any number of members can participate in the relationship type. If you specify two integers then *max* must be greater than *min*.

For example, if you specified:

```
SOURCE TYPE PROGRAM CARDINALITY 3 TO 5
TARGET TYPE ITEM
```

then only between 3 and 5 different PROGRAM members can refer to the same ITEM member via the relationship member type. You cannot validate a relationship member that does not conform to the cardinality conditions.

The CARDINALITY clause is optional.

Making Relationships via the Relationship Member Type Mandatory

To make it mandatory for members of a specified member type to participate in at least one relationship via the relationship member type, specify:

```
SOURCE TYPE member-type-list MANDATORY
TARGET TYPE member-type-list MANDATORY
```

You cannot validate a member unless it is used by a relationship member of the relationship member type being defined.

The MANDATORY clause is optional.

Controlling the Removal of Members Participating in a Relationship

By default, if you remove:

- A source or target member then:
  - The relationship member defining the relationship between the source and target is also removed
  - The remaining source or target member is marked as check needed
- A relationship member then both the source and target members are marked as check needed.

To define that all members participating in a relationship will be removed if any of the participating members are removed, specify:

```
SOURCE TYPE member-type-list CONTROLLED
TARGET TYPE member-type-list CONTROLLED
```

Controlled source and target members are removed if you remove the relationship member defining the relationship between them and they are not used by any other relationship member of the same relationship member type.

If a target member is controlled then removing the source member also removes both the target member and the relationship member. The reverse occurs if the source member is controlled and you remove the target member.

If a controlled source or target member is a relationship member that in turn references controlled members, then its removal will result in the removal of the controlled members.

The CONTROLLED clause is optional.

Allowing and Disallowing Duplicate Relationships

To allow duplicate relationships in which two or more relationship members can reference both:

- The same source member
- The same target member

specify:

```
DUPLICATES ALLOWED
```

To disallow duplicate relationships, specify:

```
DUPLICATES DISALLOWED
```

DISALLOWED prevents a relationship member being validated if it duplicates any relationship member visible in the current status window.

To allow duplicate relationships only on condition that the affected relationship member's definitions contain a user-defined attribute with a unique value, specify:

```
DUPLICATES ALLOWED DISTINGUISHED-BY attribute-type
```

where *attribute-type*:

- Is an ATTRIBUTE-TYPE member name
- Must not also be specified in the GENERIC-ATTRIBUTES clause of the MEMBER-TYPE IS RELATIONSHIP member or the COMMON-ATTRIBUTES clause of the HIERARCHY member
- Should be specified in the SEE clause but need not be specified in the ATTRIBUTES clause of the RELATIONSHIP-TYPE member

The distinguishing attribute cannot have multiple values or be mutually exclusive of another attribute in the RIM.

The DUPLICATES clause is optional. ALLOWED is the default.

Allowing a Member to be Both the Source and Target of a Relationship

To allow the same member to be both the source and target of a relationship member, specify:

```
RECURSION ALLOWED
```

To disallow recursion, specify:

```
RECURSION DISALLOWED
```

The DISALLOWED keyword prevents a relationship member being validated if the same member is both its source and target.

The RECURSION clause is optional. ALLOWED is the default.

Documenting the Order of Retrieval of Source and Target Members

Source and target members are DRETRIEVEd in random order. The ORDERED clause enables you to document alternative orders of retrieval. These alternatives are not currently supported but the clause is provided so as to enable you to document your requirements and prepare for future enhancements. The ORDERED clause is optional.

To document that retrieval will be in alphanumeric order on the source member name, specify:

```
SOURCE TYPE member-type-list ORDERED TARGET
```

To document that retrieval will be in alphanumeric order on the target member name, specify:

```
TARGET TYPE member-type-list ORDERED TARGET
```

To document that retrieval will be in alphanumeric order on the value of a user-defined attribute in the definition of the relationship member, specify:

```
SOURCE TYPE member-type-list ORDERED SEQUENCED attribute-type

TARGET TYPE member-type-list ORDERED SEQUENCED attribute-type
```

where *attribute-type*:

- Is an ATTRIBUTE-TYPE member name
- Must not also be specified in the GENERIC-ATTRIBUTES clause of the MEMBER-TYPE IS RELATIONSHIP member or the COMMON-ATTRIBUTES clause of the HIERARCHY member
- Should be specified in the SEE clause but need not be specified in the ATTRIBUTES clause of the RELATIONSHIP-TYPE member

A relationship member will not encode if its definition does not include the sequenced attribute. The sequenced attribute cannot have multiple values or be mutually exclusive of another attribute in the RIM.

To document that the sequenced attribute must contain an unique value, specify:

```
SOURCE TYPE member-type-list ORDERED SEQUENCED attribute-type
UNIQUE

TARGET TYPE member-type-list ORDERED SEQUENCED attribute-type
UNIQUE
```

A relationship member cannot be validated if another relationship member of the same relationship member type orders the same source or target member on the same attribute value.

Specifying the User-defined Attributes that can be Included in the Member Type

To define the user-defined attributes that can be included in the definition of the member type, specify:

ATTRIBUTES *attribute-list*

where *attribute-list* is:

- The names of one or more ATTRIBUTE-TYPE or ATTRIBUTE-GROUP members. All attributes defined by the named ATTRIBUTE-TYPEs or by the ATTRIBUTE-TYPEs directly and indirectly contained in the named ATTRIBUTE-GROUPs can be included in the definition of the member type. Each member name must be separated by a comma if entered via the command interface.

and/or:

- One or more sets of mutually exclusive ATTRIBUTE-TYPE members. Each set must contain two or more ATTRIBUTE-TYPE member names each separated by an ELSE keyword. The same ATTRIBUTE-TYPE member name can be specified in more than one set. Only one of the attributes defined by the alternative ATTRIBUTE-TYPE members can be present in the definition of the member type. Each set must be separated by a comma if entered via the command interface.

Each individual ATTRIBUTE-TYPE or ATTRIBUTE-GROUP member name or each set of alternative ATTRIBUTE-TYPE members can be followed with:

- OPTIONAL NO to specify that:
  - — An attribute or all the attributes contained in the named group
  - — One of the mutually exclusive attributes within a set

  must be present when a member is encoded
- OPTIONAL YES to specify that the attribute(s) need not be present
- OPTIONAL WARN to specify that the attribute(s) need not be present but, if not a warning message displays.

The default is OPTIONAL YES.

The ATTRIBUTES clause is optional.

Allowing and Disallowing Relationships Via Specified Clauses

To disallow references from the relationship member type via a particular clause, specify:

```
RELATIONSHIPS VIA clause DISALLOWED
```

where *clause* is:

- SEE
- UDR*n* (where *n* is an integer in the range 1 to 9)
- UDRS

By default, a relationship member can refer to a member of any member type via its SEE, UDR or UDRS clauses.

To restrict the member types to which the relationship member type can refer via a particular clause, specify:

```
RELATIONSHIPS VIA clause ALLOW member-name-list
```

where *member-name-list* is a list of one or more MEMBER-TYPE, MEMBER-TYPE-GROUP, RELATIONSHIP-TYPE, or RELATIONSHIP-GROUP member names. Each member name must be separated by a comma if entered via the command interface.

Only the member types defined by the named MEMBER-TYPEs and RELATIONSHIP-TYPEs, or directly and indirectly contained by the named MEMBER-TYPE-GROUPs and RELATIONSHIP-GROUPs, can be referenced via clause.

By default if a relationship member refers via a SEE, UDR, or UDRS clause to a member that does not have an encoded source record, then a dummy member having the same member type as the relationship member is created. If you specify an ALLOW clause then the member type of the dummy member is defined by the first member named in the member-name-list. If the first member is a MEMBER-TYPE-GROUP or RELATIONSHIP-GROUP, then the dummy member type is defined by the first MEMBER-TYPE or RELATIONSHIP-TYPE they contain.

To explicitly define the type of dummy member created by a reference from a relationship member via a specified clause, specify:

```
RELATIONSHIPS VIA clause DUMMY member-name
```

or

```
RELATIONSHIPS VIA clause ALLOW member-name-list DUMMY member-name
```

where *member-name* is a MEMBER-TYPE or RELATIONSHIP-TYPE member name.

The RELATIONSHIPS-VIA clause is optional.

Automatically Defining EA Relationships

To automatically define EA relationships from a member type, specify:

```
AUTO-REF-STRING string
```

where *string*:

- Is a string of up to 80 characters
- Defines an indicator which, if placed one or more character spaces before a member name in an updated member's definition, automatically creates an EA relationship between the updated and prefixed member
- Must be delimited if entered via the command interface

The EA relationship will be via whichever clause, specified in the RELATIONSHIP-TYPE member's SEE clause, is defined in an ATTRIBUTE-TYPE member having the following clauses:

```
EDIT-CODE-1 6
EDIT-CODE-2 6
SKELETON-CODE 8
```

For example, if you define a SEE string for use in HAS-ATTRIBUTES members as follows:

```
AUTO-REF-STRING SEE
```

and define that references are to be via the SEE clause:

```
ATTRIBUTE-TYPE
FREE-FORM-TEXT
IDENTIFIED-BY SEE
EDIT-CODE-1 6
EDIT-CODE-2 6
SKELETON-CODE 8
SKELETON-HELP AUTOMATICALLY MAINTAINED EA RELATIONSHIPS
```

and then specify SEE in the NOTE clause of a HAS-ATTRIBUTES member named HA-ONE:

```
NOTE
SEE HA-TWO AND SEE HA-THREE
```

then the clause:

```
SEE
HA-TWO
HA-THREE
```

is automatically generated in the definition of HA-ONE.

Only one AUTO-REF-STRING clause can be specified in a RELATIONSHIP-TYPE definition. Only one ATTRIBUTE-TYPE member in that RELATIONSHIP-TYPE's SEE clause can be defined for use by the AUTO-REF-STRING.

Any number of members can be referenced in this manner. Because these EA relationships are automatically maintained, the referenced member's names are not listed under the relationship clause in the assisted update buffer.

The AUTO-REF-STRING clause is optional.

Defining Naming Conventions for Relationship Members

To define naming conventions for relationship members, specify:

NAMING *convention-strings*

where *convention-strings* are one or more strings each containing:

- Characters from the standard and/or extended character sets for names
- Seven optional special characters, three of which are wildcards (* _ #), three of which restrict the length of the member name (> < =), and one of which allows you to incorporate variables into names (:)
- Must each be delimited if entered via the command interface

The naming conventions restrict the range of names that can be used for a member type. If, during an assisted update for a particular member type, you enter a member name that does not conform to the convention, the name is disallowed.

The naming conventions are displayed on selection list panels and enforced for updates made using both the AUPD line command and the AUPDATE primary command.

Repository member names can contain up to 32 characters from the standard or extended character sets for names. You must not define a convention which results in users attempting to create members with invalid names. Names:

- Containing only numeric characters or any character from the extended character set
- Beginning with an underscore or hyphen

must be delimited if specified in command interface functions.

The three wildcard characters you can use to specify what can appear at the corresponding positions in the member name, are as follows:

- A wildcard for open naming conventions: the position must be filled with one character. The default character is an underscore (_).
- An optional wildcard: the position can be filled with any number of characters, or none at all. The default character is an asterisk (*).
- A wildcard for numeric characters: the position must be filled with a numeric character. The default character is a hash (#).

For example, specifying the following in the definition of the RELATIONSHIP-TYPE member defining the HAS-ATTRIBUTES member type:

```
NAMING HA_#-*
```

would mean that the names of all HAS-ATTRIBUTES members must begin with HA, immediately followed by any character, then a number and then a hyphen, and ending with up to 27 alphanumeric characters, since the total maximum is 32 characters.

You can also limit the length of member names, by including any of three special characters in the NAMING clause. The characters are as follows:

- A maximum name delimiter, whose position determines the maximum allowed length of a name: the name must end before the position occupied by this character. The default character is the less than symbol (<).
- An exact length delimiter, which restricts names to a particular length: the last character of the name must immediately precede this character. The default character is the equals sign (=).
- A minimum name delimiter: the end of the name must come after the position occupied by this character. The default character is the greater than symbol (>).

The > and < symbols cannot be combined in a single naming convention.

The following are examples of how the length of member names can be restricted:

To define that a name must begin with MEM- followed by exactly 3 characters, specify:

```
NAMING MEM-___=
```

If the equals sign were not used, the name could be of any length.

To define that a name must begin with USER followed by no more than two numeric characters, specify:

```
NAMING USER##<
```

To define that a name can comprise any characters, but must be between 2 and 4 characters long, specify:

```
NAMING __**<
```

You can also include variables in the NAMING clause; at run-time, the values of the variables are included in the displayed naming convention. This allows for dynamic naming conventions, which change during a session or according to the user or date. For example, you could include a variable that contains the user's ID, so that users can only access members created by themselves.

The variable name must be enclosed by the wildcard character for the variable part of naming conventions, which defaults to a colon (:). For example, with the variable &USER set to CON, the following clause:

```
NAMING HA-:&USER:##<
```

would be, in effect:

```
NAMING HA-CON##<
```

If you want only part of the value of a variable to be included, you can specify which part, in parentheses, after the variable name:

```
NAMING :variable(first-char,length):
```

For example:

```
NAMING HA-:&USER(2,2):##<
```

would become:

```
NAMING HA-ON##<
```

with only the second and third characters of the variable &USER being included.

You can specify for a particular naming convention to be used only if a particular condition is met, by using:

NAMING IF *condition* THEN *convention-string*

where:

condition is a boolean expression.

convention-string is a naming convention string, as defined above.

For example:

```
NAMING IF &USER = MASTER THEN CO_____IMS
```

would result in the naming convention CO_____IMS only being used if the variable USER had the value MASTER. Otherwise, if this was the only NAMING clause in the RELATIONSHIP-TYPE definition, then the member type would not appear in the selection list.

Any number of conditional NAMING clauses can be included in a RELATIONSHIP-TYPE definition to cover every eventuality.

The seven special characters are set in the following variables in SEXEC EC1060:

- The > character is set in variable MDG_NAMOL
- The < character is set in variable MDG_NAMSOL
- The = character is set in variable MDG_NAMON
- The * character is set in variable MDG_NAMOPT
- The _ character is set in variable MDG_NAMJOK
- The # character is set in variable MDG_NAMNUM
- The: character is set in variable MDG_NAMVAR

You can tailor the special characters using the environment functions (provided by panel TZ42300).

The NAMING clause is optional.

Taking a User Exit Defining Complex Naming Conventions

To take a user exit defining complex naming conventions, specify:

```
NAMING-EXIT member-name
```

where *member-name*:

- Is the name of an EXECUTIVE member on the MP-AID
- Must be delimited if entered via the command interface

The user exit routine is defined in the SEXEC member from which the specified EXECUTIVE member is constructed.

The NAMING-EXIT clause is optional. If it is specified it will override any naming conventions specified in the NAMING clause.

Refer to Chapter 7, "User Exits," on page 171 for details of how to define an exit routine to check names.

Preventing a Member Type Being Displayed in the Panel Interface

To prevent a relationship member type being displayed in the panel interface, specify:

```
LONG-NAME MMR-DISAPPEAR
```

MMR-DISAPPEAR must be delimited if entered via the command interface.

The member type is included in your RIM but is not visible to users of the panel interface. For example, the member type is not displayed in selection list panels.

You therefore have the ability to continue to include obsolescent member types in your RIM.

You can still add members to the repository via the command interface using commands such as UPDATE and RESTORE.

The LONG-NAME clause is optional.

Specifying the Clauses and Keywords Displayed During Assisted Update

To define the clauses and keywords displayed during an assisted update of the relationship member type, specify:

```
SEE attribute-list
```

where *attribute-list* is a list of one or more ATTRIBUTE-TYPE and ATTRIBUTE-GROUP member names. Each member name must be separated by a comma if entered via the command interface.

Clauses and keywords are displayed in the assisted update buffer and documented in the in-context help in the order that the ATTRIBUTE-TYPE members defining them are listed in the SEE clause (whether directly or indirectly via an ATTRIBUTE-GROUP member).

If an ATTRIBUTE-GROUP is specified then the clauses and keywords are displayed in the order in which the ATTRIBUTE-TYPE members defining them are directly or indirectly contained in the group.

All clauses and keywords that you want to update via an assisted update buffer must be specified in the SEE clause.

If you have renamed UDR or UDRS clauses then the ATTRIBUTE-TYPE members in which they are defined must also be specified in the SEE clause of the HIERARCHY member containing the RELATIONSHIP-TYPE.

An ATTRIBUTE-TYPE member renaming a UDRS subclause must not be specified in a RELATIONSHIP-TYPE SEE clause. In order to display a UDRS subclause in an assisted update buffer you must specify it in the SKELETON-TEXT clause of the ATTRIBUTE-TYPE members defining the UDR clauses.

A clause that can contain free form text as its value must be specified last in the attribute-list.

The SEE clause is optional but assisted update buffers and in-context help is only available for relationship member types for which it is specified.

Defining an Alias Identifier

To define an alias identifier for the relationship member type, specify:

```
ALIAS alias
```

where *alias*:

- Is a two character string and must be unique in the Administration Repository
- Must be delimited if entered via the command interface

The alias provides a short identifier that can be used to reference the relationship member type from within macros and user exit routines.

The ALIAS clause is mandatory. You can only enable ToolSet Services if the ALIAS clause is specified.

Documenting Help for a Member Type

To document help for a relationship member type, specify:

```
HELP text
```

where *text*:

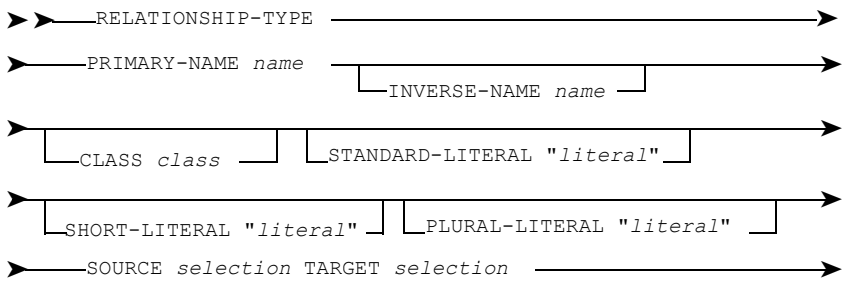
- Is free form text documenting the member type
- Is displayed as in-context help from within an assisted update buffer or in response to an MTHELP command

Help on the clauses and keywords available with the member type must be defined in the HELP clause of the relevant ATTRIBUTE-TYPE members.

The HELP clause must be the last clause specified in a RELATIONSHIP-TYPE member definition.

The HELP clause is optional.

RELATIONSHIP-TYPE Syntax



```
DUPLICATES ─┬─ALLOWED ──┬──────────────────────────┬──
            │           └─DISTINGUISHED-BY a-type ─┘
            └─DISALLOWED ──────────────────────────┘

RECURSION ─┬─ALLOWED ───┬──
           └─DISALLOWED ─┘

ATTRIBUTES ─┬──────────────────────────<<<<<<<<<<<<<<<<<<<<<<<<<<,<<<<<<<<<<<<<<<<<<
            │ a-type ─┬─<<<<<<<<<<<<<<<<<<
```

```
➤──┬──────────────
```

`executive` is an EXECUTIVE member name.

`ref-string` is a string of up to 80 characters.

`common-clauses` are any of the clauses common to all member types other than the ALIAS and SEE which are used for special purposes in the RELATIONSHIP-TYPE member type.

`help` is one or more lines of free form text.

Note:

The commas and delimiters shown in the above syntax are required when creating RELATIONSHIP-TYPE members via the command interface.

Refer to *ASG-ControlManager User's Guide* for details of the character set for names.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of the common clauses.

SEXEC

SEXEC defines an Executive Routine that can contain macros.

The important clauses of the SEXEC member type are:

- MPAID-NAME
- CONTENTS

The MPAID-NAME clause specifies the name the EXECUTIVE member constructed from the SEXEC will be given on the MP-AID. The name can be a maximum of ten characters long.

The MPAID-NAME clause is optional. The member name of the SEXEC in the Administration Repository is used as the EXECUTIVE member name if you do not specify the clause. If the SEXEC member name exceeds ten characters in length it is rejected as the EXECUTIVE name and an error message is output.

The CONTENTS clause is mandatory and specifies the sequence of instructions executed by the SEXEC. Use the following instructions for coding:

- Manager Products commands. Refer to *ASG-Manager Products Quick Reference*.
- Procedures language instructions. Refer to *ASG-Manager Products Procedures Language*
- Macros. Refer to Chapter 8, "Macros," on page 195 for details.

10 Life Cycle Services Introduction

This chapter includes these sections

Concepts 337

Benefits 340

Concepts

Life Cycle Services (LCS) is a development environment generated from Life Cycle Models.

A *Life Cycle Model* defines and documents the Application Development Life Cycle used in your company. A Life Cycle Model is much more than passive documentation. It includes a Tool Usage Model which defines the MethodManager functions and non-ASG tools used to bring the application into service. Life Cycle Models are defined in the repository in LIFE-CYCLE, PHASE, ACTIVITY, and LIFE-CYCLE-OBJECT-TYPE members.

Life Cycle Models are assigned to projects. You can create as many projects and Life Cycle Models as you want in the same repository. Different projects can share the same Life Cycle Model.

A *project* is the framework within which application development is carried out. It is a set of specific tasks that are performed within a given timeframe by relevant people for the accomplishment of a predefined goal. A predefined goal in the context of application development, is the application and its associated documentation. Each project is defined in the repository in a PROJECT member.

LCS provides a hierarchy of menus listing the phases, activities, subactivities, prerequisites, and deliverables of a project. Users assigned to a project are guided by a dialog which includes menus and in-context help in order to produce the deliverables required to complete the project.

A *phase* is the primary division of a Life Cycle Model. Each phase contains a group of related activities. The contents of a phase is determined by what you plan to achieve by using it. A phase can provide a checkpoint giving you the opportunity to:

- Ensure that all the deliverables required have been produced and are correct
- Make decisions about further development based on the progress so far achieved

Each phase is defined in the repository in a PHASE member.

A phase is broken down into a series of activities. Each activity specifies a set of *deliverables* which must be produced to complete the activity. The sequence in which they appear in the activity is the sequence in which each deliverable must be produced. Activities also specify a set of *prerequisites*. Prerequisites are deliverables that have previously been output and must now serve as input to another activity. Activities can contain *subactivities*. Each activity and subactivity is defined in the repository in an ACTIVITY member.

Selected activities and deliverables can be defined as a *task*. Tasks allow users to access the deliverables they are assigned to work on. Activities within a phase can be assigned to different tasks, but tasks must not include activities from different phases of the project. You can monitor the progress of each phase of the project by the development of its tasks. When all tasks within a phase are complete, the whole phase is complete. A task is defined in a TASK member.

Note:

Project Management functions do not support subactivities contained by sub activities

Each deliverable produced by a Life Cycle Model is known as a *Life Cycle Object*. Each Life Cycle Object is of a given type and each type is represented by a LIFE-CYCLE-OBJECT-TYPE member. A LIFE-CYCLE-OBJECT-TYPE member provides help about a deliverable and defines the procedures which drive the user through the steps required to produce a Life Cycle Object of a given type. They can also provide help about how prerequisites are used and define the procedures with which they are displayed or interrogated.

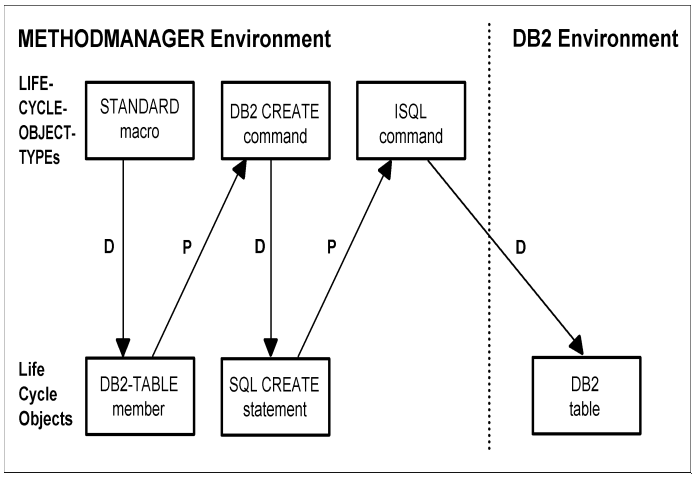
LIFE-CYCLE-OBJECT-TYPE members define the *Tool Usage Model*.

Life Cycle Models are reusable and a LIFE-CYCLE-OBJECT-TYPE member should define the procedures by which a type of Life Cycle Object can be produced for any project rather than create a specific Life Cycle Object that is particular to a single project. For example, a LIFE-CYCLE-OBJECT-TYPE member containing a STANDARD macro can be used repeatedly to create members of a particular member type whereas a LIFE-CYCLE-OBJECT-TYPE member creating a single named member can only be used once.

All deliverables must serve as a prerequisite at some time in the life cycle with the exception of the final deliverable (the application and its documentation). All prerequisites must have already been produced as a deliverable with the exception of the initial deliverables and any externally produced deliverables (for example deliverables produced by another project).

For example, the following sequence of Life Cycle Objects could be produced in order to create a DB2 table by defining it in the repository and exporting an SQL statement generated from the definition.

Figure 58 • Example LIFE-CYCLE-OBJECT-TYPEs



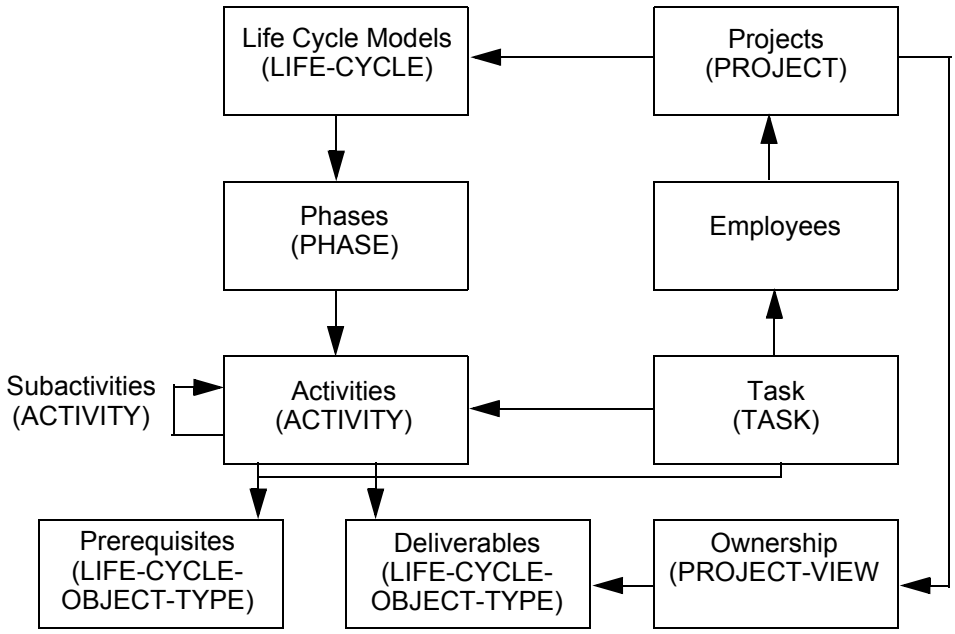
D indicates that a Life Cycle Object is a deliverable produced by a LIFE-CYCLE-OBJECT-TYPE member and **P** indicates that a Life Cycle Object is the prerequisite required before another LIFE-CYCLE-OBJECT-TYPE member can produce a further deliverable.

If a Life Cycle Object is a repository member, it is contained in the current project's *projectview* and protected against users not assigned to the project. Users not assigned to the project cannot access or update the members for as long as the project is still active. Projectviews are defined in PROJECT-VIEW members and are automatically created when you create a project using the Project Management function provided by MethodManager.

The Methodology Model Manager Method - SIP provides examples of the members defining a Life Cycle Model and project.

Refer to "Project Management" on page 378 for further details of Project Management.

Figure 59 • Defining Your Lifecycle Services Environment in the Repository. Member types are shown in upper case.



Benefits

LifeCycle Services is a development environment that provides a menu driven human-computer interface which guides users through their Application Development Life Cycle while enforcing standards.

Most companies have standards; many companies use methods; some companies have a methodology; very few have a computer-driven methodology.

In most cases, methodologies and their standards are paper-based and therefore, either not rigorous or far too tedious to be done by hand. They are theory that is rarely practiced and lies on shelves collecting dust.

LifeCycle Services enables you to take an engineering approach to systems development.

What is an engineering approach?

- It defines a methodology which integrates tried and true methods which are to be a part of the development methodology.
- It breaks down Application Development into small, definable, precise and manageable activities and then organizes the work effort needed to perform them. Each of these activities will produce a deliverable, an input to the development.
- It ensures that resources are managed so that the maximum possible benefit is squeezed out of every one of them.

Why take an engineering approach?

- Today, systems development is a complex undertaking requiring the co-ordination and integration of disparate systems, organizations and tools. You need to be able to simplify things.
- In many cases you are working with inaccurate or incomplete specifications and may not detect this fact until late in development or implementation. Systems implemented in such environments may be unreliable or not meet the information needs.
- Everything must be done within a budgetary constraint and human resource is expensive. Programming is labor intensive and affected by labor shortages.
- Despite these constraints, critical applications are needed quickly for a company to maintain its competitive edge.

What are the benefits of LifeCycle SERVICES?

- An engineering approach can be taken to application development.
- Projects are initiated, planned, and controlled.
- Projects are directed through their life cycle using the defined Life Cycle Model to guide employees through the tasks that have been assigned to them.
- It is generated from Life Cycle Models defined in the repository.
- Standards are enforced by standards definitions defined in the repository.
- Quality is controlled by automated validations at many levels. Checks are made far more consistently and rigorously than is possible by manual methods alone. The degree of rigor is determined by the Project Leader.
- Previous investment is preserved by the incorporation of tried and true methods in your Life Cycle Model.
- Most importantly, method theory is put into practice. The methodology becomes an active plan rather than a passive document.

11 Member Types Defining a Life Cycle Model

A Life Cycle Model includes:

- Phases
- Activities
- Subactivities
- Prerequisites
- Deliverables

are defined in the repository in LIFE-CYCLE, PHASE, ACTIVITY, and LIFE-CYCLE-OBJECT-TYPE members.

Assisted Update Buffers and in-context help, which describe how to create the members defining a Life Cycle Model, are provided by MethodManager.

You can display in-context help about any member type available in your repository by pressing PF1 in an Update Buffer or by using the MTHELP command.

Life Cycle Model members must be defined in the same repository as the project to which the model is assigned and in which the deliverables of the project are to be produced.

This ensures that if a deliverable is a new member, then that member is included in the project's projectview and so protected from users not assigned to the project.

Using the Change Management Function you can define Life Cycle Models and projects in a base status visible from the dependent status in which users assigned to the projects work. The deliverables of the projects are produced in the dependent status.

The STATUS PERMIT command enables you to make the base status a read-only or update status. By making the base status read-only you can prevent the members defining Life Cycle Models and projects being updated by users other than the Project Leader.

Only the Project Leader using the repository Controller's authority can create and maintain statuses.

Refer to *ASG-MethodManager Dictionary/Repository Information Model* for details of the LIFE-CYCLE, PHASE, ACTIVITY, LIFE-CYCLE-OBJECT-TYPE, and PROJECT member types.

12 Enabling Life Cycle Services

Having defined a Life Cycle Model in the repository you must enable Life Cycle Services (LCS) by generating the model so that it can be assigned to a project and made available to the users of the project.

Only the Project Leader using the System Administrator's authority can enable LCS.

Generating a Life Cycle Model constructs members onto the MP-AID from the members defining the model in the repository. The MP-AID members define:

- The hierarchy of menus that guide users through a project
- The instructions that can be executed from the Object Type Selection menu to produce the deliverables required to complete a project.

You can generate all or part of a Life Cycle Model by using the VXA or VX commands.

The VX command constructs MP-AID members from a single selected repository member.

The VXA command constructs MP-AID members from both a selected repository member and all the members that it directly and indirectly uses. For example, by applying the VXA command to a LIFE-CYCLE member you can generate an entire Life Cycle Model.

You must re-generate a Life Cycle Model if you want it to reflect any changes you have made to the repository members defining the model. Users use the last generated version of the model assigned to their project. You need re-generate only those parts of the Life Cycle Model you have changed.

You can simulate and check a Life Cycle Model before generation.

Simulation displays the menus and in-context help to be constructed but without creating MP-AID members. Simulation does not affect existing Life Cycle Models or projects.

Using the VXC command you can check before generation that a Life Cycle Model has been correctly defined in the repository and that the names of the MP-AID members to be constructed are unique.

You can also delete the MP-AID members constructed from a Life Cycle Model by using the reset functions provided by both panel TV77000 and the MP DELETE command.

The VX, VXA, VXC, AND VXP Commands

To generate part of a Life Cycle Model use the VX command. The syntax of the VX command is as follows:

```
──VX member-name ─┬─ ; ─┬──────────────────────────────────────────────►◄
                  └─ . ─┘
```

where *member-name* is the name of a LIFE-CYCLE, PHASE, ACTIVITY, or LIFE-CYCLE-OBJECT-TYPE member.

To generate all or part of a Life Cycle Model use the VXA command. The syntax of the VXA command is as follows:

```
──VXA member-name ─┬─ ; ─┬──────────────────────────────────────────────►◄
                   └─ . ─┘
```

where *member-name* is the name of a LIFE-CYCLE, PHASE, ACTIVITY, or LIFE-CYCLE-OBJECT-TYPE member.

To check before generation that a Life Cycle Model has been correctly defined in the repository and that the names of the MP-AID members to be constructed are unique within the model, use the VXC command. The syntax of the VXC command is as follows:

```
──VXC member-name ─┬─ ; ─┬──────────────────────────────────────────────►◄
                   └─ . ─┘
```

where *member-name* is the name of a LIFE-CYCLE member.

To display the executive routine generated for a Life Cycle Object Type use the VXP command. The syntax of the VXP command is as follows:

```
──VXP repository-name ─┬─ ; ─┬──────────────────────────────────────────►◄
                       └─ . ─┘
```

where *repository-name* is the name of a LIFE-CYCLE-OBJECT-TYPE member.

Note:

The VX, VXA, VXC, and VXP commands can also be specified in the line command area.

13 Instructions that Produce Deliverables or Display Prerequisites

This chapter includes these sections:

Macros 348
:CASE 348
:DCSTANDARD 349
Listing Members 351
:DISPLAY 353
:LEVEL 354
:LINE-COMMAND 354
:STANDARD 356

Commands 359
DCUPD 359
DCUPD Syntax 359
HARDCOPY 360
MATRIX 360
MTHELP 362
PROJLIST 363
PROJVIEW 366

The instructions that produce a deliverable, or display or interrogate the prerequisites required to produce a deliverable, are specified in the COMMAND attribute of LIFE-CYCLE-OBJECT-TYPE members.

The instructions can be a combination of:

- Executive routines
- Macros
- Primary and line commands
- Procedures language functions, directives, and variables.

You can call the functions provided by MethodManager and the tools provided by other vendors. You can only call other vendors tools if they can understand a Manager Products call and accept input from MethodManager.

For example, by specifying an :FMTSCREEN macro you can call the ASG-supplied export functions in order to generate SQL statements for a DB2 or SQL/DS environment.

To execute the instructions you must select the LIFE-CYCLE-OBJECT-TYPE member in which they are defined on the Object Type Selection menu.

The instructions can display several levels of output in a series of panels. For example, the first level of output could be a list of members and the second level of output a print of one of the listed members.

All the commands, executive routines, macros, functions, directives, and variables provided by Manager Products are available to users with the necessary Functional Units installed.

Refer to *ASG-Manager Products Quick Reference* for details of the instructions not documented in this publication.

Macros

This section describes the macros provided by Manager Products to support LifeCycle Services.

:CASE

The :CASE macro specifies that the instructions within a LIFE-CYCLE-OBJECT-TYPE member are only executed to produce a deliverable or to display or interrogate a prerequisite.

To define that instructions are used to produce a deliverable, specify:

```
:CASE DELIVERABLE
```

in the COMMAND attribute of a LIFE-CYCLE-OBJECT-TYPE member.

To define that the instructions are used to display or interrogate a prerequisite, specify:

```
:CASE PREREQUISITE
```

The :CASE macro is useful if the same LIFE-CYCLE-OBJECT-TYPE member defines both a deliverable and a prerequisite of an activity or subactivity.

You specify whether a LIFE-CYCLE-OBJECT-TYPE defines a prerequisite or a deliverable in the PREREQUISITE and DELIVERABLE attributes of ACTIVITY members.

For example, if you specify:

```
:LEVEL 1
 :LINE-COMMAND S
  :DISPLAY
   PROJLIST FUNCTION;
:LEVEL 2
 :LINE-COMMAND S
  :CASE PREREQUISITE
   ADISPLAY &P0;
  :CASE DELIVERABLE
   AUPD &P0;
```

then the S Line Command defined in :LEVEL 2 either prints or updates one of the listed FUNCTIONs depending on whether the S Line Command defined in :LEVEL 1 has been entered in order to display a prerequisite or to produce a deliverable.

The keyword PREREQUISITE can be replaced by the alternative keyword INPUT both in the :CASE macro and in ACTIVITY member definitions.

The keyword DELIVERABLE can be replaced by the alternative keywords OUTPUT or RESULT both in the :CASE macro and in ACTIVITY member definitions.

:CASE Syntax

```
>>──:CASE ──┬──PREREQUISITE ──┬──────────────────────────><
            ├──DELIVERABLE ───┤
            ├──INPUT ─────────┤
            ├──OUTPUT ────────┤
            └──RESULT ────────┘
```

:DCSTANDARD

The :DCSTANDARD macro lists DOCUMENT members belonging to the current project. A DOCUMENT member naming convention skeleton is also listed if the macro is specified in a LIFE-CYCLE-OBJECT-TYPE member producing a deliverable. Refer to ":DCSTANDARD Syntax" on page 353 for the syntax of the :DCSTANDARD macro.

You can:

- Update or print documentation from the listed members
- Use the skeleton to create a new member
- Display help about deliverables or prerequisites

by entering `H` , `P` or `S` Line Commands on the panel displayed by the :DCSTANDARD macro.

The effects of the S and P Line Commands vary depending on whether the :DCSTANDARD macro has been specified in a LIFE-CYCLE-OBJECT-TYPE member producing a deliverable or displaying a prerequisite.

If you are producing a deliverable:

- S opens an Assisted Update Buffer for a selected member in the same format as that opened by a DCUPD command

If you are displaying a prerequisite:

- S produces documentation from a selected member in the same format as that produced by a DOC command. The HARDCOPY command can be used to print the documentation on your printer if you have set up and initialized a Print Job

If you are producing a deliverable or displaying a prerequisite:

- H displays the help defined in the HELP attribute of the LIFE-CYCLE-OBJECT-TYPE member containing the :DCSTANDARD macro
- P produces documentation from a selected member in the same format as that produced by a DOC command

An entry is automatically inserted into the CONTENTS attribute of the DOCUMENT skeleton when you update it using an S command.

The CONTENTS is copied from the CONTENTS attribute of any existing DOCUMENT member whose name excluding its member type prefix is the same as that of the LIFE-CYCLE-OBJECT-TYPE member containing the :DCSTANDARD macro. For example, if the LIFE-CYCLE-OBJECT-TYPE member is named LT-ONE the CONTENTS is copied from any DOCUMENT member named DC-ONE.

If a DOCUMENT member with a matching name does not exist the CONTENTS is copied from the TEMPLATE attribute of the LIFE-CYCLE-OBJECT-TYPE member containing the :DCSTANDARD macro.

The CONTENTS attribute of the skeleton is not updated if a DOCUMENT member with a matching name does not exist and the LIFE-CYCLE-OBJECT-TYPE member does not have a TEMPLATE attribute.

You can therefore specify a standard format for the CONTENTS of your documents and ensure that it is used in new DOCUMENT members created using the naming convention skeleton.

Documents can be project dependent or application dependent.

Project dependent documents are those that are only relevant to the current project. ASG recommends identifying project dependent documents by adopting a naming convention in which the short name of the project to which they belong is included in the DOCUMENT member name.

The inclusion of the project's short name in DOCUMENT member names enables you to identify the project for which they were created after the project has been completed and ownership of its deliverables has been removed.

To process project dependent documents specify the keyword PROJECT in the :DCSTANDARD macro.

Application dependent documents are those that are not only relevant to the current project but may have a company wide use once the project has been completed. For example, they may form a template on which you model other documents.

To process application and project dependent documents do not specify the keyword PROJECT in the :DCSTANDARD macro.

You can adopt detailed naming conventions for both application and project dependent documents. For example, you can include a character string in the names of DOCUMENT members identifying their use. To process documents with names containing a particular character string specify a QUALIFY clause in the :DCSTANDARD macro.

Listing Members

To list all DOCUMENT members belonging to the current project, specify:

```
:DCSTANDARD
```

in the COMMAND attribute of a LIFE-CYCLE-OBJECT-TYPE member.

To list all DOCUMENT members belonging to the current project whose names include a particular character string, specify:

```
:DCSTANDARD QUALIFY string
```

For example, if you specify:

```
:DCSTANDARD QUALIFY PROBLEM-REPORT
```

then all DOCUMENT members belonging to the project with names beginning with the string DC-PROBLEM-REPORT are listed.

To list all DOCUMENT members belonging to the current project whose names include the project's short name, specify:

```
:DCSTANDARD QUALIFY PROJECT
```

For example, if the current projects short name is ECCAM then all DOCUMENT members belonging to the project with names beginning with the string DC-ECCAM are listed.

To list all DOCUMENT members belonging to the current project whose names include the project's short name followed by a character string, specify:

```
:DCSTANDARD QUALIFY PROJECT string
```

For example, if the current project's short name is ECCAM and you specify:

```
:DCSTANDARD QUALIFY PROJECT PROBLEM-REPORT
```

then all DOCUMENT members belonging to the project with names beginning with the string DC-ECCAM-PROBLEM-REPORT are listed.

To list all DOCUMENT members whose names include the project's short name prefixed by a character string, specify:

```
:DCSTANDARD QUALIFY string PROJECT
```

For example, if the current project's short name is ECCAM and you enter:

```
:DCSTANDARD QUALIFY PROBLEM-REPORT PROJECT
```

then all DOCUMENT members belonging to the project with names beginning with the string DC-PROBLEM-REPORTECCAM are listed.

A naming convention skeleton is always listed if the :DCSTANDARD macro is specified in a LIFE-CYCLE-OBJECT-TYPE member producing a deliverable. The naming convention will include the project short name and/or string if you have specified a QUALIFY clause in the :DCSTANDARD macro.

To list DOCUMENT members belonging to the current project that are kept in a named KEPT-DATA list, specify:

```
:DCSTANDARD KEPT 'KEPT IN list-name'
```

where *list-name* is the name of a KEPT-DATA list.

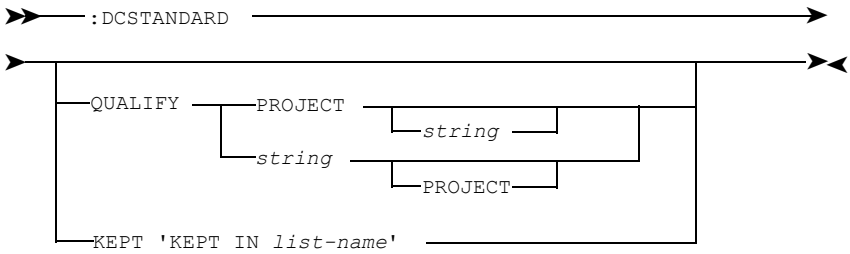
The delimiter character single quote (') can alternatively be double quote (").

For example, if you enter:

```
:DCSTANDARD KEPT "KEPT IN WORK"
```

then all the DOCUMENT members belonging to the current project that are kept in the KEPT-DATA list named WORK are listed.

:DCSTANDARD Syntax



where:

string is a character string.

list-name is the name of a KEPT-DATA list.

:DISPLAY

The :DISPLAY macro prepares output for the next level of processing by displaying it in a panel.

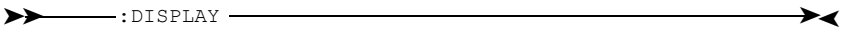
If you do not specify a :DISPLAY macro then the output of the subsequent instruction is displayed in a Lookaside Buffer. Line Commands cannot be specified in Lookaside Buffers and if you want to apply them to the output of an instruction you must specify a :DISPLAY macro.

For example, if you specify:

```
:LINE-COMMAND S
:DISPLAY
PROJLIST FUNCTION;
```

in the COMMAND attribute of a LIFE-CYCLE-OBJECT-TYPE member then the output of the PROJLIST FUNCTION command is displayed in a panel.

:DISPLAY Syntax



:LEVEL

The :LEVEL macro defines a sequence of steps by structuring instructions into separate processing levels.

To define a level, specify:

```
:LEVEL n
```

in the COMMAND attribute of a LIFE-CYCLE-OBJECT-TYPE member.

n is an integer.

For example, if you specify:

```
:LEVEL 1
 :LINE-COMMAND S
  :DISPLAY
   PROJLIST FUNCTION;
:LEVEL 2
 :LINE-COMMAND S
  :DISPLAY
   ADISPLAY &P0;
```

then the output of the PROJLIST command defined in :LEVEL 1 is displayed in a panel to which the ADISPLAY command defined in :LEVEL 2 can be applied. You could therefore print one of the listed FUNCTIONs.

By pressing PF3 you return to the panel displaying the output produced by the previous level or to the Object Type Selection menu.

:LEVEL Syntax

```
>>──:LEVEL n──────────────────────────────────────────><
```

where *n* is an integer.

:LINE-COMMAND

The :LINE-COMMAND macro defines the instructions which are executed when the Line Commands H, P, or S are applied to a LIFE-CYCLE-OBJECT-TYPE member.

To define the instructions, specify:

```
:LINE-COMMAND character
instructions
```

in the COMMAND attribute of a LIFE-CYCLE-OBJECT-TYPE member.

where:

character is H, P, or S.

instructions are a combination of any of the following:

- Executive routines
- Macros
- Primary and line commands
- Procedures language directives, functions, and variables

For example, if you specify:

```
:LINE-COMMAND S
PROJLIST FUNCTION;
```

in the COMMAND attribute of a LIFE-CYCLE-OBJECT-TYPE member and select the member using the Line Command S then all FUNCTIONs belonging to the current project are listed.

The :LINE-COMMAND macro only defines the Line Commands that are applied to the LIFE-CYCLE-OBJECT-TYPE member in which the macro is specified. Defining one character disables any other character, with the exception of H, that is applied to the member. You can define all the characters with separate :LINE-COMMAND macros.

:LINE COMMAND Syntax

```
>>──:LINE-COMMAND character instructions──────────────><
```

where:

character is H, P, or S

instructions are a combination of any of the following:

- Executive routines
- Macros
- Primary and line commands
- Procedures language directives, functions and variables

:STANDARD

The :STANDARD macro lists all members belonging to the current project that have a specified member type. Several member types can be specified. A naming convention skeleton is also listed for each member type if the :STANDARD macro is specified in a LIFE-CYCLE-OBJECT-TYPE member producing a deliverable.

You can:

- Update or print the listed members
- Use the skeletons to create new members
- Display help about the specified member types

by entering H, P, or S Line Commands on the panel displayed by the :STANDARD macro.

The effects of the S and P Line Commands vary depending on whether the :STANDARD macro has been specified in a LIFE-CYCLE-OBJECT-TYPE member producing a deliverable or displaying a prerequisite.

Refer to ":STANDARD Syntax" on page 358 for the syntax of the :STANDARD macro.

If you are producing a deliverable an S opens an Assisted Update Buffer for a selected member in the same format as that opened by an AUPD command

If you are displaying a prerequisite an S prints a selected member in the same format as that printed by an ADISPLAY command

If you are producing a deliverable or displaying a prerequisite:

- P prints a hardcopy print of a selected member in the same format as that printed by an ADISPLAY command if you have defined and initialized a Print Job
- H displays member type help in the same format as that displayed by an MTHELP command or by pressing PF1 in an Update Buffer

Listing Members

To list all members belonging to the current project that are of a specified member type, specify:

```
:STANDARD member-type-list
```

in the COMMAND attribute of a LIFE-CYCLE -OBJECT-TYPE member.

where *member-type-list* is a list of any of the member types available in your repository.

To list all members belonging to the current project whose member name includes a particular character string, specify:

```
:STANDARD member-type-list QUALIFY string
```

For example, if you specify:

```
:STANDARD ITEM QUALIFY 1
```

then all ITEM members belonging to the project whose names begin with the string IT-1 are listed.

To list all members belonging to the current project whose member name includes the project's short name, specify:

```
:STANDARD member-type-list QUALIFY PROJECT
```

For example, if the current project's short name is ECCAM and you specify:

```
:STANDARD ITEM QUALIFY PROJECT
```

then all ITEM members belonging to the project whose names begin with the string IT-ECCAM are listed.

To list all members whose member name includes the project's short name followed by a character string, specify:

```
:STANDARD member-type-list QUALIFY PROJECT string
```

For example, if the current project's short name is ECCAM and you specify:

```
:STANDARD ITEM QUALIFY PROJECT 1
```

then all ITEM members belonging to the project whose names begin with the string IT-ECCAM-1 are listed.

To list all members whose member name includes the projects short name prefixed by a character string, specify:

```
:STANDARD member-type-list QUALIFY string PROJECT
```

For example, if the current project's short name is ECCAM and you specify:

```
:STANDARD ITEM QUALIFY 1 PROJECT
```

then all ITEM members belonging to the project whose names begin with the string IT-1ECCAM are listed.

A naming convention skeleton is always listed for each of the member types in the member-type-list if the :STANDARD macro is specified in a LIFE-CYCLE-OBJECT-TYPE member producing a deliverable. The naming convention will include the project short name and/or string if you have specified a QUALIFY clause in the :STANDARD macro.

To list those members belonging to the current project that are kept in a named KEPT-DATA list, specify:

```
:STANDARD KEPT 'KEPT IN list-name'
```

where *list-name* is the name of a KEPT-DATA list you have created.

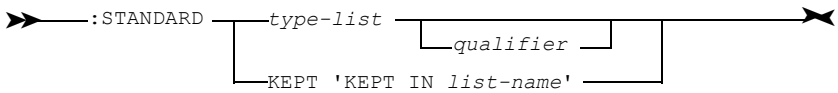
The delimiter character single quote (') can alternatively be double quote (").

For example, if you specify:

```
:STANDARD KEPT "KEPT IN WORK"
```

then all the members belonging to the current project that are kept in the KEPT-DATA list named WORK are listed.

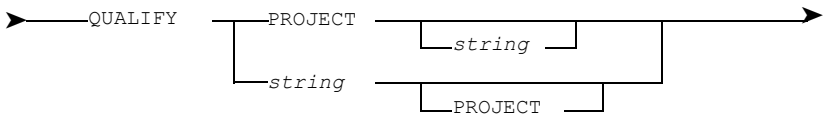
:STANDARD Syntax



where:

type-list is a list of any of the member types available your repository

qualifier is:



string is a character string.

list-name is the name of a KEPT-DATA list you have created.

Commands

This section describes the commands provided by Manager Products to support Lifecycle Services. The commands are documented in alphabetical order of command name.

DCUPD

The DCUPD command creates a new DOCUMENT member and copy into its CONTENTS attribute information from an existing member.

Enter:

```
DCUPD new-member existing-member ;
```

where:

new-member is the name of a new DOCUMENT member.

existing-member is the name of an existing DOCUMENT or LIFE-CYCLE-OBJECT-TYPE member.

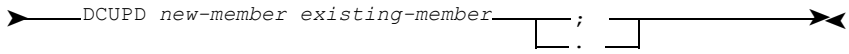
If the existing-member is a DOCUMENT member then its CONTENTS attribute is copied into the new-member.

If the existing-member is a LIFE-CYCLE-OBJECT-TYPE member then the information in its TEMPLATE clause is copied into the CONTENTS attribute of the new-member.

The DCUPD command displays the source record of the new-member in an Assisted Update Buffer.

The information is copied from the encoded record of the existing-member.

DCUPD Syntax



where:

new-member is the name of a new DOCUMENT member.

existing-member is the name of an existing DOCUMENT or LIFE-CYCLE-OBJECT-TYPE member.

HARDCOPY

The HARDCOPY command prints the contents of the current buffer on your printer.

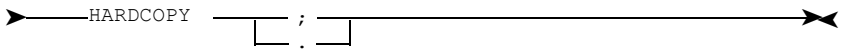
Enter:

```
HARDCOPY ;
```

Before you can use the HARDCOPY command you must create and initialize a Print Job.

You can use the HARDCOPY command to print documents generated from DOCUMENT members by the DOC command or the :DCSTANDARD macro.

HARDCOPY Syntax



MATRIX

The MATRIX command outputs a matrix report showing the relationships between members in two KEPT-DATA lists.

The members in the first KEPT-DATA list specified in the command form the horizontal axis of the matrix and the members in the second KEPT-DATA list form the vertical axis.

You can indicate on the matrix those members in the first KEPT-DATA list that:

- Directly use the members in the second KEPT-DATA list by specifying the USES keyword
- Are directly used by the members in the second KEPT-DATA list by specifying the CONSTITUTES keyword

The character X indicates that there is a direct relationship between two members via any attribute.

For example, if you enter:

```
MATRIX 'KEPT IN HORIZONTAL' USES 'KEPT IN VERTICAL' ;
```

then a matrix in the following format is reported:

```
                        AC-Y112
                        | AC-Z111
                        | | LC-ONE
                        | | | PH-Y11
AC-Z111                 . . . .
AC-Z112                 . . . .
PH-Z11                  . X . .
X = ALL
```

and if you enter:

```
MATRIX 'KEPT IN HORIZONTAL' CONSTITUTES 'KEPT IN
VERTICAL';
```

then a matrix in the following format is reported:

```
                        AC-Y112
                        | AC-Z111
                        | | LC-ONE
                        | | | PH-Y11
AC-Z111                 . . . X
AC-Z112                 . . . .
PH-Z11                  . . X .
X = ALL
```

In the above examples the members AC-Z111, AC-Z112, and PH-Z11 are in the KEPT-DATA list named HORIZONTAL and the members AC-Y112, AC-Z111, LC-ONE, and PH-Y11 are in the KEPT-DATA list named VERTICAL. PH-Z11 uses AC-Z111 and is used by LC-ONE. AC-Z111 is used by PH-Y11.

You can refine the output so that only relationships via specified attributes are displayed. If you specify particular attributes you must also specify a character with which each is represented in the matrix.

You can tailor the width of the field containing the names of the members in the first KEPT-DATA list. By default the field has a width of 32 character spaces and member names are truncated if they cannot be contained in it.

You can also tailor the spacing between columns in the matrix. By default columns are separated by one character space.

For example, if you enter

```
MATRIX 'KEPT IN HORIZONTAL' USES 'KEPT IN VERTICAL' 12 4
                CONTAINS C ;
```

then only relationships via the CONTAINS attribute (represented with the character C) are displayed on the matrix. The left margin is 12 characters wide. Columns are separated by 4 character spaces.

MATRIX Syntax

```
>>--- MATRIX 'KEPT IN list-name-1' ------------------------------>
>--+- USES -----------+--- 'KEPT IN list-name-2' ---------------->
   +- CONSTITUTES -----+

>-------------------------------------------------------------->
   +-------------------- <<<<<<<<<<<<<<<<<<<<<<<<<<<< ----------+
   +-field column attribute attribute-id ------------------------+
>--+- ; -+-------------------------------------------------------><
         +- . -+
```

where:

list-name-1 and *list-name-2* are the names of KEPT-DATA lists.

field is an integer specifying the number of character spaces in the field in which the names of the members in *list-name-1* are listed.

column is an integer specifying the number of character spaces separating columns in the matrix.

attribute is any attribute that can establish a relationship between the members in *list-name-1* and *list-name-2.*

attribute-id is a single character which represents the attribute within the matrix.

MTHELP

The MTHELP command displays help on any member type available in your repository.

To display help for a member type, enter:

MTHELP *member-type* ;

member-type is any member type available in your repository.

The text is the same as that displayed when you request help by pressing PF1 from within an Update Buffer.

MTHELP Syntax

```
>--- MTHELP member-type ---+- ; -+------------------------------><
                           +- . -+
```

where *member-type* is any member type available in your repository.

PROJLIST

The PROJLIST command lists members belonging to the current project.

Refer to "PROJLIST Syntax" on page 365 for the syntax of the PROJLIST command.

Listing Members having a Specified Member Type and Name

To list all members belonging to the current project that are of a specified member type, enter:

```
PROJLIST member-type-list ;
```

where *member-type-list* is a list of any of the member types available in your repository.

To list all members belonging to the current project having names including a specified character string, enter:

```
PROJLIST member-type-list QUALIFY string ;
```

For example, if you enter:

```
PROJLIST ITEM QUALIFY 1 ;
```

then all ITEM members belonging to the current project whose names begin with the string IT-1 are listed.

A naming convention skeleton is always listed for each of the member types in the member-type-list unless you specify the keyword NO-SKELETON in the PROJLIST command. The naming convention will include string if you specify a QUALIFY clause.

The output of the PROJLIST command displays information about the selected members in the following columns:

- Members lists all the selected members and naming convention skeletons.
- Type specifies the member type of each member.
- References specifies the number of members that directly use each member.
- Condition specifies whether each member is CONSISTENT (encoded), INCONSISTENT (unverified) or RESERVED (dummy).
- Access specifies whether you have the authority to read, update or alter each member.

Listing Members with Names Including the Project Name

To list all members belonging to the current project whose member name includes the project's short name, enter:

```
PROJLIST member-type-list QUALIFY PROJECT ;
```

where *member-type-list* is a list of any of the member types available in your repository.

For example, if the current project's short name is ECCAM and you enter:

```
PROJLIST DOCUMENT QUALIFY PROJECT;
```

then all DOCUMENT members belonging to the project whose names begin with DC-ECCAM are listed.

To list all members whose member name includes the project's short name followed by a specified character string, enter:

```
PROJLIST member-type-list QUALIFY PROJECT string ;
```

For example, if the current project's short name is ECCAM and you enter:

```
PROJLIST DOCUMENT QUALIFY PROJECT 1 ;
```

then all DOCUMENT members belonging to the project whose names begin with the string DC-ECCAM-1 are listed.

To list all members whose member name includes the projects short name prefixed by a specified character string, enter:

```
PROJLIST member-type-list QUALIFY string PROJECT ;
```

For example, if the current project's short name is ECCAM and you enter:

```
PROJLIST DOCUMENT QUALIFY 1 PROJECT;
```

then all DOCUMENT members belonging to the project whose names begin with the string DC-1ECCAM are listed.

A naming convention skeleton including the project short name and string is always listed for each of the member types in the member-type-list unless you specify the NO-SKELETON keyword.

Listing Members in a Named KEPT-DATA List

To list those members belonging to the current project that are kept in a named KEPT-DATA list, enter:

```
PROJLIST KEPT 'KEPT IN list-name' ;
```

where *list-name* is the name of a KEPT-DATA list you have created.

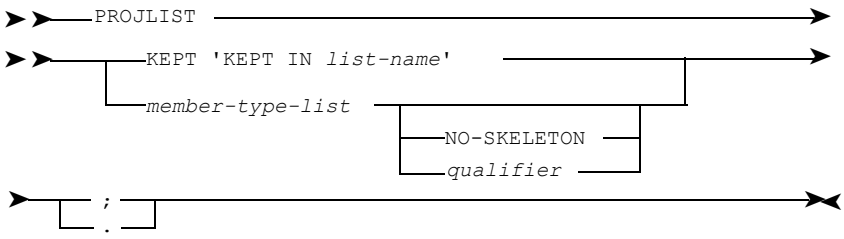
The delimiter character single quote (') can alternatively be double quote (").

For example, if you enter:

```
PROJLIST KEPT "KEPT IN WORK" ;
```

then all the members belonging to the current project that are kept in the KEPT-DATA list named WORK are listed.

PROJLIST Syntax

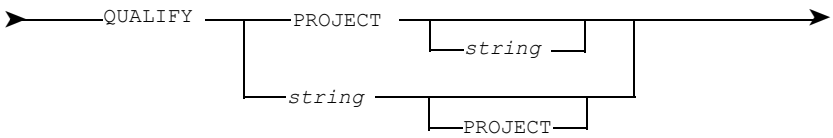


where:

list-name is the name of a KEPT-DATA list you have created.

member-type-list is a list of any of the member types available your repository.

qualifier is:



where *string* is any character string.

PROJVIEW

The PROJVIEW command creates a KEPT-DATA list containing all the members of a specified member type that belong to the current project.

To keep a list of members, enter:

```
PROJVIEW member-type list-name ;
```

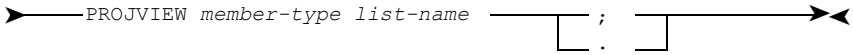
where:

member-type is any member type available in your repository.

list-name is the name of the KEPT-DATA list.

If *list-name* already exists its contents will be replaced by the output of the latest PROJVIEW command.

PROJVIEW Syntax



where:

member-type is any member type available in your repository.

list-name is the name of the KEPT-DATA list.

14 An Example of a Life Cycle Model

This chapter includes these sections:

An Example of a Phase 368
An Example of an Activity 369
An Example of a Subactivity 370
An Example of a LIFE-CYCLE-OBJECT-TYPE 371

The LIFE-CYCLE member LC-MMR-SIP naming and defining the structure of the Life Cycle Model MMR-SIP:

Figure 60 • Example of a LIFE-CYCLE Member

```
 LIFE-CYCLE
OPTION 'MMR-SIP' ------------------------------ ---> A
 OPTION-TEXT
'METHODMANAGER Strategic Information Planning'  ---> B
 HELP
'METHODMANAGER ...' --------------------------- ---> C
 EFFECTIVE-DATE 01.06.90
 CONTAINS
 PH-M21
 ,PH-M22
 ,PH-M23
 ,PH-M24
 ,PH-M25
```

A and B are the name and description identifying the Life Cycle Model on the LifeCycle Services panels from which models are generated, simulated or displayed, and C is help that can be displayed by entering the Line Command H in front of A on these panels.

An Example of a Phase

The PHASE member PH-M21 defining the Gaining Commitment phase of the Life Cycle Model MMR-SIP:

Figure 61 • Example of a PHASE Member

```
PHASE
OPTION '1' ---------------------- ----------------------------+
OPTION-TEXT 'Gaining Commitment' ----------------------------+
CONTAINS                                                     :
AC-M211                                                      :
,AC-M212                                                     :
,AC-M213                                                     :
,AC-M214                                                     :
,AC-M215                                                     :
,AC-M216                                                     :
EDITOR 'JDC'                                                 :
EFFECTIVE-DATE '01.07.89'                                    :
HELP 'This first phase ...' ---- ----------------------------:---> A
                                                             :
                                                             :
The generated phase menu:                                    :
                                                             :
 ----> Continue with line command  s=selection, h=help       :
                                                             :
Project: Eccam International Suits Corporation - SIP Study  -:---> B
Phase:   Menu                                                :
Activity:                                                    :
------------------------------------------------------------ :
===== 1        Gaining Commitment <------------------------ -+
===== 2        Short Strategy Study
===== 3        Full Strategy Study
===== 4        Tactical Implementation Planning
===== 5        Ongoing Update
```

A is help that can be displayed by entering the Line Command H in front of the OPTION on the phase menu.

B is the name of the project to which the Life Cycle Model is assigned. If the Life Cycle Model has not yet been assigned to a project, for example during simulation, then the model name (in these examples MMR-SIP) is displayed.

An Example of an Activity

The ACTIVITY member AC-M211 defining the Establish Scope of Study activity of the phase Gaining Commitment:

Figure 62 • Example of an ACTIVITY Member

```
ACTIVITY
CONTAINS
SU-M2111
,SU-M2112
,SU-M2113
,SU-M2114
HELP 'Commitment...' ----------------- -------------------------------> A
OPTION '1' --------------------------- -----------------------+
OPTION-TEXT 'Establish Scope Of Study' -----------------------+
                                                               :
                                                               :
   Generated activity menu:                                    :
                                                               :
   ----> Continue with line command  s=selection, h=help       :
                                                               :
   Project:  Eccam International Suits Corporation - SIP Study :
   Phase:    Gaining Commitment ------------------------------ -:----> B
   Activity: Menu                                              :
  ------------------------------------------------------------ :
  ===== 1        Establish Scope Of Study  <------------------ -+
  ===== 2        Define The Study Roles
  ===== 3        Develop Schedule For Study
  ===== 4        Make A Presentation To Management
  ===== 5        Review Study Proposals
  ===== 6        Confirm Commitment
```

A is help that can be displayed by entering the Line Command H in front of the OPTION on the activity menu.

B is the name of the phase the activities are in.

An Example of a Subactivity

The ACTIVITY member SU-M2111 defining the subactivity Establish Business Areas of the activity Establish Scope of Study:

Figure 63 • Example of a Subactivity

```
ACTIVITY
EDITOR 'DH'
HELP '!Goal...' ---------------------- ----------------------------> A
OPTION '1' --------------------------- -------------------------+
INPUTS                                                          :
LT-NO-PREREQUISITE                                              :
OUTPUTS                                                         :
LT-BUSINESS-AREAS-ESTABLISHED                                   :
OPTION-TEXT 'Establish Business Areas' -------------------------:
                                                                :
                                                                :
Generated subactivity menu:                                     :
                                                                :
----> Continue with line command  s=selection, h=help           :
                                                                :
Project:    Eccam International Suits Corporation - SIP Study   :
Phase:      Gaining Commitment                                  :
Activity:   Establish Scope Of Study ------------------------- -:---> B
Subactivity:Menu                                                :
-------------------------------------------------------------   :
===== 1     Establish Business Areas <------------------------- -+
===== 2     Set Goals For Study
===== 3     Develop Business Reasons
===== 4     Perform Risk Analysis
```

A is help that can be displayed by entering the Line Command H in front of the OPTION on the subactivity menu.

B is the name of the activity the subactivities are in.

An Example of a LIFE-CYCLE-OBJECT-TYPE

The LIFE-CYCLE-OBJECT-TYPE member LT-BUSINESS-AREAS-ESTABLISHED defining the deliverable Business Areas Established of the subactivity Establish Business Areas:

Figure 64 • Example of a LIFE-CYCLE-OBJECT-TYPE Member

```
LC-OBJECT-TYPE
HELP '!Goal...' ------------------- --------------------------------> A
OPTION 'Business Areas Established' ----------------------------+
OPTION-TEXT ' '                                                 :
TYPE Optional --------------------- ----------------------------+
TOOL ' ' -------------------------- ----------------------------+
TEMPLATE 'layout'                                               :
COMMAND                                                         :
:DCSTANDARD DC-QU901169-TEST                                    :
*  :LINE-COMMAND S                                              :
*     LOOK PANEL M2111;                                         :
                                                                :
                                                                :
Generated Object Type Selection menu:                           :
                                                                :
----> Continue with line command  s=select, h=help, p=print     :
                                                                :
Project:    Eccam International Suits Corporation - SIP Study   :
Phase:      Gaining Commitment                                  :
Activity:   Establish Scope of Study                            :
Subactivity:Establish Business Areas -------------------------- -:---> B
------------------------------------------------------------    :
===== Prerequisite                 M/O        Tool              :
===== ---------------------------- ---------- ----------------  :
===== None                         Optional   None              :
===== ---------------------------- ---------- ----------------  :
===== Deliverable                  M/O        Tool              :
===== ---------------------------- ---------- ----------------  :
===== Business Areas Established   Optional        <----------- -+
```

A is help that can be displayed by entering the Line Command H in front of the OPTION on the subactivity menu.

B is the name of the subactivity for which the deliverable is to be produced.

15 Producing Documentation

You can produce documents by:

- Defining them as the deliverables of a Life Cycle Model
- Using the examples provided by LifeCycle Services
- Using the documentation functions provided by ToolSet Services
- Using the DOC, DCUPD, and HARDCOPY commands.

Documents are defined in the repository in DOCUMENT members.

You can produce documents as the deliverables of Life Cycle Models by specifying the following instructions in LIFE-CYCLE-OBJECT-TYPE members:

- :DCSTANDARD, DCUPD, DOC, and HARDCOPY to create DOCUMENT members and print hardcopy documentation from them
- :FMTSCREEN to call the documentation functions.

You can use the DOC, DCUPD, and HARDCOPY commands and the documentation functions directly, that is independently of a project or Life Cycle Model, in order to create, interrogate, or print documentation.

To get hardcopy printouts of your documentation you must first use the environment functions (menu Z71000) to set up and initialize a print job.

16 Procedures for Creating and Maintaining Life Cycle Models

MethodManager provides functions for creating and maintaining Life Cycle Models.

A Life Cycle Model produces a network of Life Cycle Objects as its deliverables. Each deliverable output by an activity must also form a prerequisite input to another activity later in the project. Ensure that this network is complete when creating a Life Cycle Model and that it is not disrupted by any changes later made to the model.

Be careful when changing a Life Cycle Model assigned to a currently active project particularly if the project has already passed the phase which you want to change.

For example, if you include a new member type in the RIM you must define a LIFE-CYCLE-OBJECT-TYPE member to create whatever members of that member type are required to complete the project. The impact of including the LIFE-CYCLE-OBJECT-TYPE member in your Life Cycle Model must be carefully assessed.

These procedures for creating or maintaining Life Cycle Models are supported:

Determining the Life Cycle Model members and projects affected by a change. Use the repository functions or the US, USA, USR, REF, and REFA Line Commands to find out the members that use or are used by the members you are changing.

Copying Life Cycle Models. Copying a model retains a copy of it in its original form to return to if your updates are abandoned. Copying an existing model is also a fast way of creating a new one.

Creating or updating Life Cycle Model members. Assisted update buffers help you to create or change the members defining a Life Cycle Model.

Testing Life Cycle Models for consistency. You can display the prerequisites input to and the deliverables output by each activity or subactivity.

Simulating Life Cycle Models. Simulation displays a model as it will be generated and allows you to check the model and any changes you may have made to it.

Generating Life Cycle Models. Generating a new or changed model constructs it onto the MP-AID and makes it available to the users of the project to which the model is assigned.

Documenting Life Cycle Models. DOCUMENT members should be updated to reflect any changes you make to a model.

17 Project Management: Interactive Functions

This chaper includes these sections:

Project Management 378
Create a Project 379
Assign Existing User to Project 379
Add and Assign New User 380
Exclude User From Project 381
Delete User From Repository 381
Delete Project From Repository 381
List all Users 382
List Projects Visible From the Current Status 382
List all Projects 382
Task Management 383
Select Project 383
Monitor Project Development 384
Select Project 385
Monitor Task Development 385
Review of Project Members 386
Remove Dummies From Project-View 386
Include Project Related Members in Project-View 386

Project Management

The functions listed on this menu allow a Project Leader, with the repository Controller's authority, to create and maintain projects and assign or remove users from projects.

To ensure that a project's development follows a defined methodology, the Project Leader must assign a Life Cycle Model to a project.

If you want to work on the phases and activities in a project, return to the top level menu and select the Project Work function.

Create
Use this function to create a new project and automatically protect it.

Assign Model
Use this function to assign a Life Cycle Model to a project.

Assign User
Use this function to give a user a security level to access a project. The user must have a name and password in the repository.

Add
Use this function to give a new user a name and password in the repository and a security level to access a project.

Exclude
Use this function to exclude a user from a project by removing their security level to access the project.

Delete User
Use this function to delete a user's name and password from the repository.

Delete Project
Use this function to delete a project from the repository.

List Users
Use this function to list all the users with a name and password in the repository.

List Visible
Use this function to list all the projects visible from the current status.

List All
Use this function to list all projects, to check that they have an entry in the repository and are defined as an owner.

Task
Use this function to list, create, update and monitor tasks.

Review
Use this function to add members to, or remove members from, the current project's project-view.

User
The User defined option allows your Systems Administrator to insert further options in the menu.

Create a Project

Use this function to create a new project. The project will automatically be protected.

Explanation of the input fields:

Project short name
To create a new project, enter a name of up to 6 characters without the member type prefix.

Project long name
To give the project a meaningful, longer title, enter a name of up to 50 characters.

The long name is used as the project title in the Life Cycle Model menus displayed when a user is working on a project. If you do not enter a long name, the short name is used as the project title.

Assign Existing User to Project

Use this function to provide an existing user with a security level to access a project. The user must have a name and password in the repository. A user can be assigned to more than one project.

Explanation of the input fields:

User name
To specify the repository user to be assigned to a project, enter the user's name.

Project
To assign the repository user to a specific project, enter the project short name, without the member type prefix.

Security level

To specify a user's access level to a project, enter one of these security levels:

READ	Allows the user to see the members protected by a project
UPDATE	Allows the user to update the members protected by a project
ALTER	Allows the user to update and remove the members protected by a project
CONTROL	Allows the user to update and remove the members protected by a project

In a forthcoming release, CONTROL will give the user the repository Controller's authority, which allows access to all Project Management functions.

Add and Assign New User

Use this function to provide a new user with a name and password in the repository and assign to them a security level to access a project.

To assign a user to a project if they already have a name and password in the repository, use the Assign User function.

To ascertain that a user has a name and password in the repository, use the List User function.

Explanation of the input fields:

User name

To add a new user to the repository, enter the name of the user. Use a hyphen or underscore to represent a space between two parts of a name, for example: John_Smith

This means that when you select the name from a list, it is recognized as one string.

Password

To give a new user an authority recognized by the repository, enter a password.

Project

To assign a new user to a specific project, enter the project's short name, without the member type prefix.

Security level

To specify a user's access level to a project, enter one of these security levels:

READ	Allows the user to see the members protected by a project
UPDATE	Allows the user to update the members protected by a project
ALTER	Allows the user to update and remove the members protected by a project
CONTROL	Allows the user to update and remove the members protected by a project.

Exclude User From Project

Use this function to remove a user's security level from a project. This will prevent the user from accessing the project.

Explanation of the input fields:

User name

To specify the user to be excluded from a project, enter the user's name.

Project

To specify the project from which the user is to be excluded, enter the project short name, without the member type prefix.

Delete User From Repository

The user's name and password are deleted from the repository when you enter `s` in the Line Command Area beside the user's name.

Delete Project From Repository

You can only delete a project if:

- No members are owned by the project
- No user is assigned to the project
- No other member refers to the project

To delete a project enter `s` in the Line Command Area to the left of the project name.

List all Users

All the users with a user name and password in the repository are listed.

To display all the projects to which a user is assigned, enter **s** in the Line Command Area beside the user's name.

List Projects Visible From the Current Status

All the projects visible from the current status are listed in alphabetical order.

The display includes these details:

- Member type
- The number of members that use the project
- The condition of the member; if it is an encoded, source, or dummy entry
- Security levels and protection

To display all the users assigned to a project, enter **s** in the Line Command Area beside the project name.

List all Projects

Use this selection panel to check that all projects in the repository are consistent.

All projects from every status are listed. If the project is encoded and is defined as an owner, the field to the right of the project name is blank.

The message `No Repository Entry` appears beside the project name if the member is not in a visible status.

The message No Security Entry appears beside the project name if the member is a dummy member or is not defined as an owner.

To display all the users assigned to a project, enter `s` in the Line Command Area beside the project name.

Task Management

The functions listed on this menu allow you to list, create and update tasks and evaluate their progress.

List. Use this function to list the tasks within a project.

Update. Use this function to create new tasks or update the information in existing tasks.

Monitor. Use this function to monitor the development of a specific task and the activities and LIFE-CYCLE-OBJECT-TYPE members assigned to the task.

User. The User defined option allows your Systems Administrator to insert further options on the menu.

You can also create, update, and evaluate tasks using the Work on a project function. The following ACTIVITY and LIFE-CYCLE-OBJECT-TYPE members must be included in the Life Cycle Model assigned to the active project:

AC-LCS-TASK-1	LT-LCS-PROJECT
AC-LCS-TASK-2	LT-LCS-TASK
AC-LCS-TASK-3	LT-LCS-TASK-STATE

Select Project

A list of projects displays. To list all the TASK members defined for a project, enter `s` beside the project name. A list of all tasks defined for that project is then displayed.

Monitor Project Development

Use this panel to monitor the development of tasks within the active project.

Explanation of the input fields:

Project. To select tasks from a specific project, enter the project short name. This is the only mandatory entry.

Standing. To select tasks from all standings of development leave this field blank. To select tasks from a specific standings of development, enter one of these:

- D stands for Draft
- P stands for Proposed
- A stands for Approved
- S stands for Standardized

User. To select tasks assigned to a specific user, enter the repository user's name.

Planned end after. To select the tasks planned to end on or after a specific date, enter the date.

Planned end before. To select the tasks planned to end on or before a specific date, enter the date.

Completed after. To select the tasks completed on or after a specific date, enter the date.

Completed before. To select the tasks completed on or before a specific date, enter the date.

Select Project

A list of projects is displayed.

To list all the TASK members defined for a project, enter `s` in the Line Command Area beside the project name. A list of all tasks defined for that project is displayed.

Monitor Task Development

Use this panel to monitor the development of tasks within a selected project.

Explanation of the input fields:

All phases (Y/N). To select tasks from all phases of the active project enter `Y`. To select tasks only from the current phase enter `N`. This is the only mandatory entry.

Standing. To select tasks from all standings of development leave this field blank. To select tasks from a specific standing of development, enter one of these:

- D stands for Draft
- P stands for Proposed
- A stands for Approved
- S stands for Standardized

User. To select tasks assigned to a specific user, enter the repository user's name.

Planned end after. To select the tasks planned to end on or after a specific date, enter the date.

Planned end before. To select the tasks planned to end on or before a specific date, enter the date.

Completed after. To select the tasks completed on or after a specific date, enter the date.

Completed before. To select the tasks completed on or before a specific date, enter the date.

Review of Project Members

The functions of this menu enable you to list dummies/members of a project and to deactivate/activate them in the project-view.

Deactivate. Use the first function to list or deactivate all dummies which are only included in the project-view but not used by the current or the specified project.

Activate. Use the second function to list or activate all members which are used by the current or the specified project but not included in its project-view.

Remove Dummies From Project-View

Use this function to list or remove the dummy members that are still included in the project-view of the specified project but are no longer used by the project.

Explanation of the input fields:

Short name of project. To specify the project whose project-view you want to check for dummy members, enter the short name of the project. If you make no entry, the current project is taken by default.

Dummies: list/remove. You can either list or remove dummy members. To list all dummies in the project-view, enter X in the list field. To remove all dummies in the project-view, enter X in the remove field.

Include Project Related Members in Project-View

Use this function to list or include the members that are used by the specified project, but are not included in its project-view.

Explanation of the input fields:

Short name of project. To check for members that are not included in the project view of a project, enter the short name of the project. If you make no entry, the current project is taken by default.

Project related members: list/include. You can either list or include members. To list members not included in the project-view, enter X in the list field, or to include members into the project-view, enter X in the include field.

18 PROJECT-VIEW Members

PROJECT-VIEW members are generated and maintained automatically by MethodManager. Because of this, users need not necessarily be aware of their existence. The following brief description has been included because their presence does sometimes affect Systems Administrators.

Functions

The function of PROJECT-VIEW members is to determine which repository member definitions are within the scope of a project. They operate through the Project Management Security system. For each project there is one main PROJECT-VIEW member with one subsidiary PROJECT-VIEW member for each of the member types defined within the scope of the project.

Naming Conventions

The main PROJECT-VIEW member is named as follows:

`PT-`*`name`*

where *`name`* is the project's short-name.

The subsidiary PROJECT-VIEW member relating to individual member types have the same name as the main PROJECT-VIEW member except that the appropriate member type prefix is appended to the name as a suffix. For example, the PROJECT-VIEW member relating to ITEM members would be named as follows:

`PT-`*`name`*`-IT`

where *`name`* is the project's short-name.

IMPACT ANALYSIS & PROJECT-VIEW: MDG_PROJECTVIEW_RUCOUNTS

This variable controls the output from MMR Impact Analysis functions (the US, REF, USA, and REFA commands) and is relevant to users of the LifeCycle Services Project Management Functions.

Possible settings are Y and N. The default setting is N. The output from US, REF, REFA, and USA commands will ignore any PROJECT-VIEW member-type information, but the output includes reference or usage counts which may include PROJECT-VIEWs.

If it is desirable to count any occurrences of PROJECT-VIEW member-types in this context and output an additional line advising how many have been ignored during impact analysis, choose the Y option.

Appendix A
Listing of RIM Definition

Here is a full listing of the RIM definition used in "Example of Implementing a RIM" on page 26.

```
REPLACE UH-EXAMPLE.
HIERARCHY
MP-AID-NAME "TEST1"
CONTAINS
MMG-EXAMPLE
RELATIONSHIPS
              MRT-PERSON-REPORTS-TO-PERSON ,
              MRT-PERSON-CODES-PROGRAM ,
              MRT-SYSTEM-CONTAINS-PROGRAM ,
              MRT-PROGRAM-CONTAINS-SUB ,
              MRT-SUB-CALLS-SUB ,
              MRT-S-CALLS-S-PASSING-GROUP.
REPLACE MMG-EXAMPLE.
MEMBER-TYPE-GROUP
OPTION "1"
OPTION-NAME  "EXAMPLE"
OPTION-TEXT "MODELING PROGRAM DEVELOPMENT"
CONTAINS
              MMT-PERSON ,
              MMT-PROGRAM ,
              MMT-SYSTEM ,
              MMT-SUB ,
              MMT-GROUP ,
              MMT-ITEM
SEE
              MMT-PERSON ,
              MMT-PROGRAM ,
              MMT-SYSTEM ,
              MMT-SUB ,
              MMT-GROUP ,
              MMT-ITEM ,
              MRT-SUB-CALLS-SUB ,
              MRT-S-CALLS-S-PASSING-GROUP ,
              MRT-PERSON-REPORTS-TO-PERSON ,
              MRT-PERSON-CODES-PROGRAM ,
              MRT-SYSTEM-CONTAINS-PROGRAM ,
              MRT-PROGRAM-CONTAINS-SUB
.
REPLACE MRT-PERSON-REPORTS-TO-PERSON.
RELATIONSHIP-TYPE
PRIMARY-NAME REPORTS-TO
INVERSE-NAME SUPERVISED-BY
NAMING "RE-#>"
```

```
ALIAS "RE"
SHORT-LITERAL "REPORTS-TO"
SOURCE TYPE MMT-PERSON
   CARDINALITY MANY
TARGET TYPE MMT-PERSON
   CARDINALITY 1
RECURSION DISALLOWED
RELATIONSHIPS VIA UDR1 DISALLOWED
RELATIONSHIPS VIA UDR2 DISALLOWED
RELATIONSHIPS VIA UDR3 DISALLOWED
RELATIONSHIPS VIA UDR4 DISALLOWED
RELATIONSHIPS VIA UDR5 DISALLOWED
RELATIONSHIPS VIA UDR6 DISALLOWED
RELATIONSHIPS VIA UDR7 DISALLOWED
RELATIONSHIPS VIA UDR8 DISALLOWED
RELATIONSHIPS VIA UDR9 DISALLOWED
RELATIONSHIPS VIA UDRS DISALLOWED
RELATIONSHIPS VIA SEE DISALLOWED
SEE MAT-SOURCE
  , MAT-TARGET
HELP
A REPORTS-TO member documents a person reporting to a person by defining a
source and a target PERSON member.
.
REPLACE MRT-PERSON-CODES-PROGRAM.
RELATIONSHIP-TYPE
PRIMARY-NAME CODES
INVERSE-NAME CODED-BY
NAMING "CO-#>"
ALIAS "CO"
SHORT-LITERAL "CODES"
SOURCE TYPE MMT-PERSON
   CARDINALITY 1
TARGET TYPE MMT-PROGRAM
   CARDINALITY MANY
   MANDATORY
SEE MAT-SOURCE
  , MAT-TARGET
RELATIONSHIPS VIA UDR1 DISALLOWED
RELATIONSHIPS VIA UDR2 DISALLOWED
RELATIONSHIPS VIA UDR3 DISALLOWED
RELATIONSHIPS VIA UDR4 DISALLOWED
RELATIONSHIPS VIA UDR5 DISALLOWED
RELATIONSHIPS VIA UDR6 DISALLOWED
RELATIONSHIPS VIA UDR7 DISALLOWED
RELATIONSHIPS VIA UDR8 DISALLOWED
RELATIONSHIPS VIA UDR9 DISALLOWED
RELATIONSHIPS VIA UDRS DISALLOWED
RELATIONSHIPS VIA SEE DISALLOWED
HELP
A CODES member documents a person coding a program by defining a source PERSON
member and a target PROGRAM member.
.

REPLACE MRT-SYSTEM-CONTAINS-PROGRAM.
RELATIONSHIP-TYPE
PRIMARY-NAME SYSTEM-CONTAINS-PROGRAM
INVERSE-NAME PROGRAM-CONTAINED-BY-SYSTEM
NAMING "SCP-#>"
ALIAS "SP"
SHORT-LITERAL "SYS-C-PROG"
SOURCE TYPE MMT-SYSTEM
```

```
   CARDINALITY 1
TARGET TYPE MMT-PROGRAM
   CARDINALITY MANY
CLASS CONTAINS
SEE MAT-SOURCE
  , MAT-TARGET
RELATIONSHIPS VIA UDR1 DISALLOWED
RELATIONSHIPS VIA UDR2 DISALLOWED
RELATIONSHIPS VIA UDR3 DISALLOWED
RELATIONSHIPS VIA UDR4 DISALLOWED
RELATIONSHIPS VIA UDR5 DISALLOWED
RELATIONSHIPS VIA UDR6 DISALLOWED
RELATIONSHIPS VIA UDR7 DISALLOWED
RELATIONSHIPS VIA UDR8 DISALLOWED
RELATIONSHIPS VIA UDR9 DISALLOWED
RELATIONSHIPS VIA UDRS DISALLOWED
RELATIONSHIPS VIA SEE DISALLOWED
HELP
A SYSTEM-CONTAINS-PROGRAM member documents a system containing a program by
defining a source SYSTEM member and a target PROGRAM member.
.
REPLACE MRT-PROGRAM-CONTAINS-SUB.
RELATIONSHIP-TYPE
PRIMARY-NAME PROGRAM-CONTAINS-SUBROUTINE
INVERSE-NAME SUBROUTINE-CONTAINED-BY-PROGRAM
ALIAS 'PC'
NAMING "PCS-#>"
SHORT-LITERAL "PROG-C-SUB"
SOURCE TYPE MMT-PROGRAM
   CARDINALITY MANY
TARGET TYPE MMT-SUB
   CARDINALITY MANY
   CONTROLLED
DUPLICATES DISALLOWED
CLASS CONTAINS
SEE MAT-SOURCE
  , MAT-TARGET
RELATIONSHIPS VIA UDR1 DISALLOWED
RELATIONSHIPS VIA UDR2 DISALLOWED
RELATIONSHIPS VIA UDR3 DISALLOWED
RELATIONSHIPS VIA UDR4 DISALLOWED
RELATIONSHIPS VIA UDR5 DISALLOWED
RELATIONSHIPS VIA UDR6 DISALLOWED
RELATIONSHIPS VIA UDR7 DISALLOWED
RELATIONSHIPS VIA UDR8 DISALLOWED
RELATIONSHIPS VIA UDR9 DISALLOWED
RELATIONSHIPS VIA UDRS DISALLOWED
RELATIONSHIPS VIA SEE DISALLOWED
HELP
A PROGRAM-CONTAINS-SUBROUTINE member documents a program containing a
subroutine by defining a source PROGRAM member and a target SUBROUTINE member.
.
REPLACE MRT-SUB-CALLS-SUB.
RELATIONSHIP-TYPE
PRIMARY-NAME CALLS
INVERSE-NAME CALLED-BY
ALIAS "CA"
NAMING "CA-#>"
SHORT-LITERAL "SUB-C-SUB"
SOURCE TYPE MMT-SUB
   CARDINALITY MANY
TARGET TYPE MMT-SUB
```

```
   CARDINALITY MANY
SEE MAT-SOURCE
  , MAT-TARGET
RELATIONSHIPS VIA UDR1 DISALLOWED
RELATIONSHIPS VIA UDR2 DISALLOWED
RELATIONSHIPS VIA UDR3 DISALLOWED
RELATIONSHIPS VIA UDR4 DISALLOWED
RELATIONSHIPS VIA UDR5 DISALLOWED
RELATIONSHIPS VIA UDR6 DISALLOWED
RELATIONSHIPS VIA UDR7 DISALLOWED
RELATIONSHIPS VIA UDR8 DISALLOWED
RELATIONSHIPS VIA UDR9 DISALLOWED
RELATIONSHIPS VIA UDRS DISALLOWED
RELATIONSHIPS VIA SEE DISALLOWED
HELP
A CALLS member documents a subroutine calling a subroutine by defining a source
and a target SUBROUTINE member.
.
REPLACE MRT-S-CALLS-S-PASSING-GROUP.
RELATIONSHIP-TYPE
PRIMARY-NAME PASSING
INVERSE-NAME PASSED-BY
ALIAS "PA"
NAMING "PA-#>"
SHORT-LITERAL "PASSING"
SOURCE TYPE MRT-SUB-CALLS-SUB
   CARDINALITY MANY
TARGET TYPE MMT-GROUP
   CARDINALITY MANY
ATTRIBUTES MAT-PARAMETER-NUMBER
SEE MAT-SOURCE
  , MAT-TARGET
  , MAT-PARAMETER-NUMBER
RELATIONSHIPS VIA UDR1 DISALLOWED
RELATIONSHIPS VIA UDR2 DISALLOWED
RELATIONSHIPS VIA UDR3 DISALLOWED
RELATIONSHIPS VIA UDR4 DISALLOWED
RELATIONSHIPS VIA UDR5 DISALLOWED
RELATIONSHIPS VIA UDR6 DISALLOWED
RELATIONSHIPS VIA UDR7 DISALLOWED
RELATIONSHIPS VIA UDR8 DISALLOWED
RELATIONSHIPS VIA UDR9 DISALLOWED
RELATIONSHIPS VIA UDRS DISALLOWED
RELATIONSHIPS VIA SEE DISALLOWED
HELP
A PASSING member documents a subroutine passing a parameter to a subroutine by
defining a source CALLS member and a target GROUP member.
.
REPLACE MMT-PERSON.
MEMBER-TYPE
BASED-ON ITEM
ALIAS "PE"
STANDARD-LITERAL "PERSON"
ENCODE-KEYWORD PERSON
NAMING "PE-**"
SEE MAT-DEPT
  , MAT-JOB-TITLE
  , MAT-NAME
ATTRIBUTES MAT-DEPT
  , MAT-JOB-TITLE
  , MAT-NAME
RELATIONSHIPS VIA UDR1 DISALLOWED
```

```
RELATIONSHIPS VIA UDR2 DISALLOWED
RELATIONSHIPS VIA UDR3 DISALLOWED
RELATIONSHIPS VIA UDR4 DISALLOWED
RELATIONSHIPS VIA UDR5 DISALLOWED
RELATIONSHIPS VIA UDR6 DISALLOWED
RELATIONSHIPS VIA UDR7 DISALLOWED
RELATIONSHIPS VIA UDR8 DISALLOWED
RELATIONSHIPS VIA UDR9 DISALLOWED
RELATIONSHIPS VIA UDRS DISALLOWED
RELATIONSHIPS VIA SEE DISALLOWED
RELATIONSHIPS VIA USER-EXIT DISALLOWED
RELATIONSHIPS VIA NAME DISALLOWED
RELATIONSHIPS VIA IF DISALLOWED
HELP
A PERSON member documents an employee of your organization.
.
REPLACE MAT-PARAMETER-NUMBER.
ATTRIBUTE-TYPE
INTEGER
IDENTIFIED-BY PARAMETER-NUMBER
SKELETON-HELP "Any Integer Greater Than One"
HELP
An integer greater than one defining the sequence in which parameters are passed.
.
REPLACE MAT-DEPT.
ATTRIBUTE-TYPE
CHARACTER-STRING
IDENTIFIED-BY DEPARTMENT
HELP
The department in your organization in which a PERSON works.
.
REPLACE MAT-JOB-TITLE.
ATTRIBUTE-TYPE
CHARACTER-STRING
IDENTIFIED-BY JOB-TITLE
HELP
The job title of a PERSON.
.
REPLACE MAT-NAME.
ATTRIBUTE-TYPE
CHARACTER-STRING
IDENTIFIED-BY NAME
HELP
The name of a person in your organization.
.
REPLACE MMT-PROGRAM.
MEMBER-TYPE
IS PROGRAM
ALIAS "PR"
NAMING "PR-**"
STANDARD-LITERAL "PROGRAM"
ENCODE-KEYWORD PROGRAM
SEE MAT-LANGUAGE
  , MAT-DATE-WRITTEN
RELATIONSHIPS VIA UDR1 DISALLOWED
RELATIONSHIPS VIA UDR2 DISALLOWED
RELATIONSHIPS VIA UDR3 DISALLOWED
RELATIONSHIPS VIA UDR4 DISALLOWED
RELATIONSHIPS VIA UDR5 DISALLOWED
RELATIONSHIPS VIA UDR6 DISALLOWED
RELATIONSHIPS VIA UDR7 DISALLOWED
RELATIONSHIPS VIA UDR8 DISALLOWED
```

```
RELATIONSHIPS VIA UDR9 DISALLOWED
RELATIONSHIPS VIA UDRS DISALLOWED
RELATIONSHIPS VIA SEE DISALLOWED
RELATIONSHIPS VIA INPUTS DISALLOWED
RELATIONSHIPS VIA OUTPUTS DISALLOWED
RELATIONSHIPS VIA UPDATES DISALLOWED
RELATIONSHIPS VIA PARAMETERS DISALLOWED
RELATIONSHIPS VIA PASSING DISALLOWED
RELATIONSHIPS VIA CONTAINS DISALLOWED
RELATIONSHIPS VIA CALLS DISALLOWED
RELATIONSHIPS VIA QUALIFIED-ON DISALLOWED
RELATIONSHIPS VIA ACCESS DISALLOWED
RELATIONSHIPS VIA GIVING DISALLOWED
RELATIONSHIPS VIA GIVING-THROUGH DISALLOWED
RELATIONSHIPS VIA GIVING-(THROUGH) DISALLOWED
RELATIONSHIPS VIA EDIT-NAME DISALLOWED
RELATIONSHIPS VIA COUNTS-AS DISALLOWED
RELATIONSHIPS VIA SELECTING DISALLOWED
RELATIONSHIPS VIA USER-PASSWORD DISALLOWED
RELATIONSHIPS VIA GIVING-IN DISALLOWED
RELATIONSHIPS VIA COMMBLOCK-MEMBER DISALLOWED
RELATIONSHIPS VIA SOURCE-SSR DISALLOWED
RELATIONSHIPS VIA TARGET-SSR DISALLOWED
RELATIONSHIPS VIA VIEWS DISALLOWED
HELP
A PROGRAM member documents a set of actions or instructions that a machine is
capable of executing as a whole.
.
REPLACE MMT-SYSTEM.
MEMBER-TYPE
IS SYSTEM
ALIAS "SY"
STANDARD-LITERAL "SYSTEM"
ENCODE-KEYWORD SYSTEM
NAMING "SY-**"
SEE MAT-LANGUAGE
  , MAT-DATE-WRITTEN
RELATIONSHIPS VIA UDR1 DISALLOWED
RELATIONSHIPS VIA UDR2 DISALLOWED
RELATIONSHIPS VIA UDR3 DISALLOWED
RELATIONSHIPS VIA UDR4 DISALLOWED
RELATIONSHIPS VIA UDR5 DISALLOWED
RELATIONSHIPS VIA UDR6 DISALLOWED
RELATIONSHIPS VIA UDR7 DISALLOWED
RELATIONSHIPS VIA UDR8 DISALLOWED
RELATIONSHIPS VIA UDR9 DISALLOWED
RELATIONSHIPS VIA UDRS DISALLOWED
RELATIONSHIPS VIA SEE DISALLOWED
RELATIONSHIPS VIA INPUTS DISALLOWED
RELATIONSHIPS VIA OUTPUTS DISALLOWED
RELATIONSHIPS VIA UPDATES DISALLOWED
RELATIONSHIPS VIA PARAMETERS DISALLOWED
RELATIONSHIPS VIA PASSING DISALLOWED
RELATIONSHIPS VIA CONTAINS DISALLOWED
RELATIONSHIPS VIA CALLS DISALLOWED
RELATIONSHIPS VIA QUALIFIED-ON DISALLOWED
RELATIONSHIPS VIA ACCESS DISALLOWED
RELATIONSHIPS VIA GIVING DISALLOWED
RELATIONSHIPS VIA GIVING-(THROUGH) DISALLOWED
RELATIONSHIPS VIA GIVING-THROUGH DISALLOWED
RELATIONSHIPS VIA EDIT-NAME DISALLOWED
RELATIONSHIPS VIA COUNTS-AS DISALLOWED
```

```
RELATIONSHIPS VIA SELECTING DISALLOWED
RELATIONSHIPS VIA USER-PASSWORD DISALLOWED
RELATIONSHIPS VIA GIVING-IN DISALLOWED
RELATIONSHIPS VIA COMMBLOCK-MEMBER DISALLOWED

RELATIONSHIPS VIA SOURCE-SSR DISALLOWED
RELATIONSHIPS VIA TARGET-SSR DISALLOWED
RELATIONSHIPS VIA VIEWS DISALLOWED
HELP
A SYSTEM documents a set of manual and automated procedures which work together
to satisfy one or more of the information needs of the organization.
.
REPLACE MMT-SUB.
MEMBER-TYPE
IS MODULE
ALIAS "SU"
STANDARD-LITERAL "SUBROUTINE"
ENCODE-KEYWORD SUBROUTINE
NAMING "SU-**"
SEE MAT-LANGUAGE
  , MAT-DATE-WRITTEN
RELATIONSHIPS VIA UDR1 DISALLOWED
RELATIONSHIPS VIA UDR2 DISALLOWED
RELATIONSHIPS VIA UDR3 DISALLOWED
RELATIONSHIPS VIA UDR4 DISALLOWED
RELATIONSHIPS VIA UDR5 DISALLOWED
RELATIONSHIPS VIA UDR6 DISALLOWED
RELATIONSHIPS VIA UDR7 DISALLOWED
RELATIONSHIPS VIA UDR8 DISALLOWED
RELATIONSHIPS VIA UDR9 DISALLOWED
RELATIONSHIPS VIA UDRS DISALLOWED
RELATIONSHIPS VIA SEE DISALLOWED
RELATIONSHIPS VIA INPUTS DISALLOWED
RELATIONSHIPS VIA OUTPUTS DISALLOWED
RELATIONSHIPS VIA UPDATES DISALLOWED
RELATIONSHIPS VIA PARAMETERS DISALLOWED
RELATIONSHIPS VIA PASSING DISALLOWED
RELATIONSHIPS VIA CONTAINS DISALLOWED
RELATIONSHIPS VIA CALLS DISALLOWED
RELATIONSHIPS VIA QUALIFIED-ON DISALLOWED
RELATIONSHIPS VIA ACCESS DISALLOWED
RELATIONSHIPS VIA GIVING DISALLOWED
RELATIONSHIPS VIA GIVING-THROUGH DISALLOWED
RELATIONSHIPS VIA GIVING-(THROUGH) DISALLOWED
RELATIONSHIPS VIA EDIT-NAME DISALLOWED
RELATIONSHIPS VIA COUNTS-AS DISALLOWED
RELATIONSHIPS VIA SELECTING DISALLOWED
RELATIONSHIPS VIA USER-PASSWORD DISALLOWED
RELATIONSHIPS VIA GIVING-IN DISALLOWED
RELATIONSHIPS VIA COMMBLOCK-MEMBER DISALLOWED
RELATIONSHIPS VIA SOURCE-SSR DISALLOWED
RELATIONSHIPS VIA TARGET-SSR DISALLOWED
RELATIONSHIPS VIA VIEWS DISALLOWED
HELP
A SUBROUTINE member documents a subroutine, a component of a program.
.

REPLACE MMT-GROUP.
MEMBER-TYPE
IS GROUP
ALIAS "GR"
STANDARD-LITERAL "GROUP"
```

```
ENCODE-KEYWORD GROUP
NAMING "GR-**"
SEE MAT-CONTAINS
RELATIONSHIPS VIA UDR1 DISALLOWED
RELATIONSHIPS VIA UDR2 DISALLOWED
RELATIONSHIPS VIA UDR3 DISALLOWED
RELATIONSHIPS VIA UDR4 DISALLOWED
RELATIONSHIPS VIA UDR5 DISALLOWED
RELATIONSHIPS VIA UDR6 DISALLOWED
RELATIONSHIPS VIA UDR7 DISALLOWED
RELATIONSHIPS VIA UDR8 DISALLOWED
RELATIONSHIPS VIA UDR9 DISALLOWED
RELATIONSHIPS VIA UDRS DISALLOWED
RELATIONSHIPS VIA SEE DISALLOWED
RELATIONSHIPS VIA USER-EXIT DISALLOWED
RELATIONSHIPS VIA IF DISALLOWED
RELATIONSHIPS VIA KEYS DISALLOWED
RELATIONSHIPS VIA BOUND DISALLOWED
RELATIONSHIPS VIA CONTAINS ALLOW MMT-ITEM, MMT-GROUP
HELP
A GROUP defines a group of ITEM and GROUP members.
.
REPLACE MMT-ITEM.
MEMBER-TYPE
IS ITEM
ALIAS "IT"
STANDARD-LITERAL "ITEM"
ENCODE-KEYWORD ITEM
NAMING "IT-**"
SEE MAT-HELD-AS
  , MAT-ENTERED-AS
  , MAT-REPORTED-AS
  , MAT-DEFAULTED-AS
RELATIONSHIPS VIA UDR1 DISALLOWED
RELATIONSHIPS VIA UDR2 DISALLOWED
RELATIONSHIPS VIA UDR3 DISALLOWED
RELATIONSHIPS VIA UDR4 DISALLOWED
RELATIONSHIPS VIA UDR5 DISALLOWED
RELATIONSHIPS VIA UDR6 DISALLOWED
RELATIONSHIPS VIA UDR7 DISALLOWED
RELATIONSHIPS VIA UDR8 DISALLOWED
RELATIONSHIPS VIA UDR9 DISALLOWED
RELATIONSHIPS VIA UDRS DISALLOWED

RELATIONSHIPS VIA SEE DISALLOWED
RELATIONSHIPS VIA USER-EXIT DISALLOWED
RELATIONSHIPS VIA NAME DISALLOWED
RELATIONSHIPS VIA IF DISALLOWED
HELP
An ITEM documents a fundamental element of data, the smallest named unit into
which data is divided in an organization.
.
REPLACE CONTAINS.
RELATIONSHIP-CLASS
PRIMARY-NAME CONTAINS
INVERSE-NAME CONTAINED-BY
.
REPLACE MAT-LANGUAGE.
ATTRIBUTE-TYPE
CHARACTER-STRING
IDENTIFIED-BY LANGUAGE
SKELETON-HELP "Which computer language?"
```

```
HELP
The name of a computer language.
.
REPLACE MAT-DATE-WRITTEN.
ATTRIBUTE-TYPE
DATE
IDENTIFIED-BY DATE-WRITTEN
SKELETON-HELP "Date from which the member is effective"
SKELETON-CODE 4
HELP
Contains a date, in your organization's standard format, stating when this
member was set up. It is needed if a COBOL source program is generated from
the member. Otherwise it is used for general documentation.
.
REPLACE MAT-CONTAINS.
ATTRIBUTE-TYPE
TEXT
IDENTIFIED-BY CONTAINS
SKELETON-HELP "Subordinate GROUPs and ITEMs"
EDIT-CODE-1 3
EDIT-CODE-2 7
SKELETON-TEXT
"&P2"
"ELSE &P2"
"&P2"
HELP
List the names of GROUP and ITEM members contained in this GROUP.
.

REPLACE MAT-HELD-AS.
ATTRIBUTE-TYPE
TEXT
IDENTIFIED-BY HELD-AS
REPEAT-CODE M
SKELETON-HELP "The held form of the item. Select one"
EDIT-CODE-1 4
EDIT-CODE-2 1
SKELETON-TEXT
"&P2 CHARACTER      &P2"
"&P2 BINARY         &P2"
"&P2 ALPHABETIC     &P2"
"&P2 ALPHANUMERIC   &P2"
"&P2 NUMERIC        &P2"
"&P2 PACKED-DECIMAL &P2"
"&P2 HEXADECIMAL    &P2"
HELP
The form in which the item is held in the computer. A maximum of 15 HELD-AS
clauses can be defined.
.
REPLACE MAT-ENTERED-AS.
ATTRIBUTE-TYPE
TEXT
IDENTIFIED-BY ENTERED-AS
REPEAT-CODE M
SKELETON-HELP "The form of the item on entry. Select one"
EDIT-CODE-1 4
EDIT-CODE-2 1
SKELETON-TEXT
"&P2 CHARACTER      &P2"
"&P2 BINARY         &P2"
"&P2 ALPHABETIC     &P2"
"&P2 ALPHANUMERIC   &P2"
```

```
"&P2 NUMERIC        &P2"
"&P2 PACKED-DECIMAL &P2"
"&P2 HEXADECIMAL    &P2"
HELP
The form of the item when it is entered in the computer. A maximum of 15
ENTERED-AS clauses can be defined.
.
REPLACE MAT-REPORTED-AS.
ATTRIBUTE-TYPE
TEXT
IDENTIFIED-BY REPORTED-AS
REPEAT-CODE M
SKELETON-HELP "The reported form of the item. Select one"
EDIT-CODE-1 4
EDIT-CODE-2 1
SKELETON-TEXT
"&P2 CHARACTER      &P2"
"&P2 BINARY         &P2"
"&P2 ALPHABETIC     &P2"
"&P2 ALPHANUMERIC   &P2"
"&P2 NUMERIC        &P2"
"&P2 PACKED-DECIMAL &P2"
"&P2 HEXADECIMAL    &P2"
HELP
The form in which the item is output. A maximum of 15 REPORTED-AS clauses can
be defined.
.
REPLACE MAT-DEFAULTED-AS.
ATTRIBUTE-TYPE
TEXT
IDENTIFIED-BY DEFAULTED-AS
SKELETON-HELP "The default form of the item. Select one"
EDIT-CODE-1 4
EDIT-CODE-2 1
SKELETON-TEXT
"CHARACTER      &P2"
"BINARY         &P2"
"ALPHABETIC     &P2"
"ALPHANUMERIC   &P2"
"NUMERIC        &P2"
"PACKED-DECIMAL &P2"
"HEXADECIMAL    &P2"
HELP
The form of an item that may relate to either input, internal processing or
output. Only one DEFAULTED-AS clause can be defined.
.
REPLACE MAT-SOURCE.
ATTRIBUTE-TYPE
NAME
IDENTIFIED-BY
SOURCE
EDIT-CODE-1 1
EDIT-CODE-2 1
NOTE "WAS TEXT"
```

```
SKELETON-HELP "Source of relationship"
HELP
The name of the member which is the source of the relationship.
.
REPLACE MAT-TARGET.
ATTRIBUTE-TYPE
NAME
IDENTIFIED-BY TARGET
EDIT-CODE-1 1
EDIT-CODE-2 1
SKELETON-HELP "Target of relationship"
HELP
The name of the member which is the target of the relationship.
.
```

Appendix B
Superseded Macros

Use of the following macros should be discontinued, because their functionality has been replaced by Manager Products procedures language directives.

The macros are documented in alphabetic order of macro name.

:DO FOR

The :DO FOR macro builds up a loop with a specified number of repetitions.

For example, if you specify:

```
:DO FOR MDL_COUNT FROM MDL_START TO MDL_STOP
instruction
:ENDDO
```

the variable MDL_COUNT is initialized with the start value set in MDL_START. The instructions to be executed in this loop are repeated as long as the value of MDL_COUNT is less than or equal to the value of MDL_STOP. The end of the instructions is indicated by the :ENDDO macro. MDL_COUNT is incremented by 1 with each successive iteration. Execution stops when the value of MDL_COUNT is greater than the value of MDL_STOP.

For example, if you specify:

```
:DO FOR MDL_COUNT FROM 1 TO 120 BY 2
instruction
:ENDDO
```

the variable MDL_COUNT is set to 1. The following instructions are executed and MDL_COUNT is incremented by 2 with each successive iteration. Execution stops when the value of MDL_COUNT exceeds 120.

The :DO FOR macro is superseded by the DO directive.

Refer to *ASG-Manager Products Procedures Language* for details of the DO directive.

:DO FOR Syntax

```
➤➤──:DO FOR variable FROM start TO end ──────────────────────────➤
                                        └─BY step ─┘
```

where:

variable is any local, global, or command variable that serves as a counter

start is any local, global, or command variable or a numeric value that initializes the counter

end is any local, global, or command variable or a numeric value that specifies the termination criterion of the loop

step is any local, global, or command variable or a numeric value that specifies the value by which the counter is to be incremented with each successive iteration.

:DO FOREVER

The :DO FOREVER macro builds up an unconditional loop.

For example, if you specify:

```
:DO FOREVER
instruction
:LEAVE
```

the following instructions are executed until the loop is terminated by the :LEAVE macro.

Note:

If you build up an unconditional loop using :DO FOREVER, the :LEAVE macro must be used to terminate the loop.

The :DO FOREVER macro is superseded by the DO directive.

Refer to *ASG-Manager Products Procedures Language* for details of the DO directive.

:DO WHEN

The :DO WHEN macro specifies a condition that must be valid if the following instructions are to be executed.

For example, if you specify:

```
:DO WHEN MDL_DAY EQ MDL_END_MONTH
instruction
:ENDDO
```

the following instructions are executed if the value in MDL_DAY is equal to the value set in MDL_END_MONTH. The end of the instructions is indicated by the :ENDDO macro.

For example, if you specify:

```
:DO WHEN MDL_DAY = 29 OR MDL_DAY = 30 OR MDL_DAY = 31
instruction
:ENDDO
```

the following instructions are executed if the value in MDL_DAY is equal to 29, 30, or 31. The end of the instructions is indicated by the :ENDDO macro.

The :DO WHEN macro is superseded by the IF directive.

Refer to *ASG-Manager Products Procedures Language* for details of the IF directive.

:DO WHEN Syntax

```
>>──:DO WHEN clause──┬──────────────────────────────────┬──────────────><
                     │<<<<<<<<<<<<<<<<<<                │
                     ├─AND ──────clause─────────────────┤
                     ├─OR ───────clause─────────────────┤
                     └─XOR ──────clause─────────────────┘
```

where *clause* is *variable1 comparison-operator variable2*.

where:

variable1 is a local, global, or command variable.

comparison-operator is one of the following:

EQ =
GE >=
GT >
LE <=
LT <
NE

variable2 is a local, global, or command variable or an alphanumeric value.

:DO WHILE

The :DO WHILE macro builds up a loop which iterates until the defined condition is invalid.

For example, if you specify:

```
:DO WHILE MDL_COUNT LT MDL_MAX
instruction
:ENDDO
```

the following instructions are executed as long as the value in MDL_COUNT is less than the value set in MDL_MAX. The end of the instructions is indicated by the :ENDDO macro.

For example, if you specify:

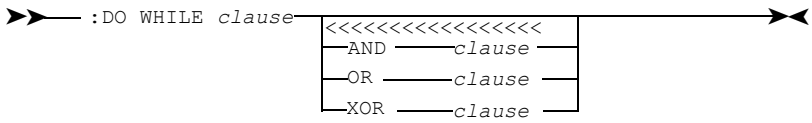
```
:DO WHILE MDL_COUNT < 33 AND MDL_COUNT > 2
instruction
:ENDDO
```

the following instructions are executed if the value in MDL_COUNT is less than 33, and greater than 2. The end of the instructions is indicated by the :ENDDO macro.

The :DO WHILE macro is superseded by the DO directive.

Refer to *ASG-Manager Products Procedures Language* for details of the DO directive.

:DO WHILE Syntax



where *clause* is *variable1 comparison-operator variable2.*

where *variable1* is a local, global, or command variable.

comparison-operator is one of the following:

EQ =
GE >=
GT >
LE <=
LT <
NE

variable2 is a local, global, or command variable or an alphanumeric value.

:ELSE

The :ELSE macro indicates the beginning of instructions following the ELSE-branch in an IF-condition.

The :ELSE macro must only be used in conjunction with the :IF macro.

The :ELSE macro is superseded by the IF directive.

Refer to *ASG-Manager Products Procedures Language* for details of the IF directive.

:ENDDO

The :ENDDO macro indicates the end of an instruction block, following a :DO ... macro.

You can have an unlimited number of instruction blocks, each delimited by :DO ... and :ENDDO. The instruction blocks can be nested to any depth.

Example:

```
:DO WHEN MDL_DAY EQ 15
    :DO FOR MDL_COUNT FROM MDL_START TO MDL_STOP
    instruction
    :ENDDO
:ENDDO
```

The :ENDDO macro must be used in conjunction with the :DO FOR, :DO WHEN, and :DO WHILE macro.

The :ENDDO macro is superseded by the DO directive.

Refer to *ASG-Manager Products Procedures Language* for details of the DO directive.

:ENDIF

The :ENDIF macro indicates the end of an instruction block, following an :ELSE macro, and indicates the end of the preceding IF-condition.

You can have an unlimited number of instruction blocks, each delimited by :IF and :ENDIF. The instruction blocks can be nested to any depth.

Example:

```
:IF MDL_FIELD1 = MDL_COMP THEN DO
instruction
:ELSE
    :IF MDL_FIELD2 = MDL_COMP THEN DO
    instruction
    :ELSE
        :IF MDL_FIELD3 = MDL_COMP THEN DO
        instruction
        :ENDIF
    :ENDIF
:ENDIF
```

The :ENDIF macro must be used in conjunction with the :IF macro.

The :ENDIF macro is superseded by the IF directive.

Refer to *ASG-Manager Products Procedures Language* for details of the IF directive.

:IF

The :IF macro specifies a condition. If the condition is valid, the instructions following the THEN-branch are executed. If the condition is invalid, the instructions following the ELSE-branch are executed.

For example, if you specify:

```
:IF MDL_FIELD1 EQ MDL_FIELD2 THEN DO
instruction
:ELSE
instruction
:ENDIF
```

the instructions following the THEN-branch will be executed, if the condition which is specified in the IF-clause is valid. Otherwise the THEN-branch will be ignored, and processing continues with the instructions following the ELSE-branch.

Note:

You need not specify the keyword DO in an :IF macro, if instructions directly follow on the keyword THEN. See example:

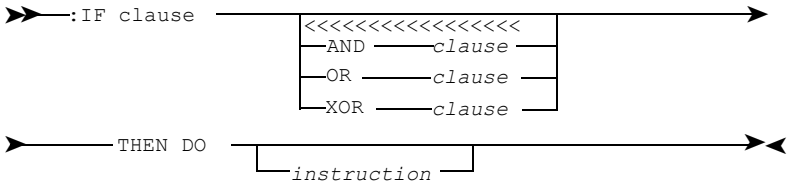
```
:IF MDL_FIELD1 = MDL_FIELD2 OR MDL_FIELD3 > 0 THEN instruction
:ELSE ...
:ENDIF
```

To increase readability of the code, ASG recommends you use a separate line for each instruction following the THEN DO and :ELSE.

The :IF macro is superseded by the IF directive.

Refer to *ASG-Manager Products Procedures Language* for details of the IF directive.

:IF Syntax



where:

clause is *variable1 comparison-operator variable2*.

where:

variable1 is a local, global, or command variable.

comparison-operator is one of the following:

EQ =
GE >=
GT >
LE <=
LT <
NE

variable2 is a local, global, or command variable or an alphanumeric value.

instruction is a single command or directive or a series of commands and directives

:LEAVE

The :LEAVE macro indicates the position where the processing of instructions is interrupted in a block delimited by :DO ... and :ENDDO. The process continues with the first instruction that follows the :ENDDO macro.

For example, if you specify:

```
:DO ...
instruction
    :IF ...
    instruction
    :ELSE
        :LEAVE
    :ENDIF
:ENDDO
SET ...
```

the processing of instructions will be interrupted, if the condition specified in the statement of the :IF macro is not true. The process will continue with the SET directive that follows the :ENDDO macro.

If several instruction blocks are nested, :LEAVE refers to the instruction block of the directly preceding :DO ... macro.

The :LEAVE macro is superseded by the LEAVE directive.

Refer to *ASG-Manager Products Procedures Language* for details of the LEAVE directive.

:LOOP

The :LOOP macro sets up a loop for an instruction block, delimited by :DO ... and :ENDDO. If several instruction blocks are nested, :LOOP refers to the instruction block of the directly preceding :DO ... macro.

For example, if you specify:

```
:DO WHEN ...
instruction
    :DO FOR ...
    instruction
        :LOOP
    :ENDDO
instruction
:ENDDO
```

the :LOOP macro refers to the preceding :DO FOR macro. :LOOP causes the repetition of the instructions that follow the :DO FOR macro as long as the :DO FOR statement is true.

The :LOOP macro is superseded by the ITERATE directive.

Refer to *ASG-Manager Products Procedures Language* for details of the ITERATE directive.

Index

Symbols

:BROWSE macro 198
:CASE macro 348
:DO FOR macro 401
:DO FOREVER macro 402
:DO WHEN macro 403
:DO WHILE macro 404
:ELSE macro 405
:ENDDO macro 405
:ENDIF macro 405
:IF macro 406
:INCLUDE macro 200
:LEAVE macro 408
:LEVEL macro 354
:LOOP macro 408
:LPARM macro 200
:NAMKO macro 201
:NAMKOT macro 206
:OUTE macro 207
:RETAIN macro 208
    customizing 140
:VCHNG macro 209
:VSEARCH macro 210

A

ACTIVITY 343
ACTIVITY member type 338
ALIAS checking 90
AMEND command
    bypassing 117
assisted update 117
    customizing 109, 175, 191
    example 33
    global exit routines 175
    help definition 61
    help generation 90
    local exit routines 191
    return to panel interface 177
    skeleton definition 61
    skeleton generation 91
attribute 18
attribute type 18
ATTRIBUTE-GROUP member type 218
ATTRIBUTE-TYPE member type 220
Automatic checking 39

C

Cardinality property 11
check-needed members 39
check-ok members 39
Class of relationship type 43
Clause 18
    type 18
Cluster menu
    definition 61
    example 33
command interface customization 118
Compare RIM 91
Construct RIM 91
contextual help 68
Control RIM 91
Control UDR 91
Controlled property 13
conventions page xv
customizing
    Life Cycle Services (LCS) 151
customizing the environment 105
CX command 87

D

DCUPD 359
deliverable 338
disabling an environment 101
DISPLAY 353
DOC command 373
documentation function customization 129
dummies property 14
DUMMY attribute type checking 90
duplicates property 15
Dynamic exit 193

E

EA prefixed EXECUTIVE members 99
EA relationship 17
    type 17
EASY-USER member 107
EC0955 SEXEC 173
EC0995 SEXEC 184
EC1060 SEXEC 107
EC9900 SEXEC 173, 182
EC9901 SEXEC 185, 188
EC9910 SEXEC 173, 182

EC9920 SEXEC 173, 182
EC9940 SEXEC 173, 177
EC9949 SEXEC 173, 177
EC9960 SEXEC 185
EC9970 SEXEC 185–186
EC9971 SEXEC 185, 187
EC9972 SEXEC 185, 187
EC9974 SEXEC 185, 188
EC9980 SEXEC 173–174, 189
EC9981 SEXEC 173, 175, 189
EC9991 SEXEC 173, 177, 191
EC9992 SEXEC 173, 177, 191
EC9993 SEXEC 173, 177, 191
EC9994 SEXEC 173, 177
EC9995 SEXEC 173, 180, 192
EC9996 SEXEC 173, 180, 192
EC9997 SEXEC 173, 180, 192
EC9998 SEXEC 173, 180
EC9999 SEXEC 173, 181
EH8000 SEXEC 107
ENCODE-KEYWORDS
    checking 90
entity
    member 17
    member type 17
    type 8
environment
    enabling 89
ER relationship 17
    type 17
EW3510 SEXEC 189, 192
EW3520 SEXEC 189, 192
EXECUTIVE members 99
extended help 68

F

file process
    tailoring 178, 192
FMTOUT command 82
FMT-SCREEN member type 247

G

global exit routines 172
global variables 106
    user-defined 106
GR-MMR-USER member 106

H

HARDCOPY 360
HDS table 22
HDS-TABLE
    generation 92
    syntax 270
HDS-TABLE member type 268
help panel
    defining 68
HIERARCHY member type 271
    syntax 279

I

INFOBANK-PANEL member type 282
input panel definition 51
Integrity checking 39
Interrogation 43
ITEM member type 282
    syntax 285

L

Life Cycle
    tailoring assignment to project 187
Life Cycle Model 337, 343
    tailoring panel display 185
Life Cycle Object 338
Life Cycle Services
    assigning user to project 187
    customizing 151
        member type clauses 154
        member type relationships 158
        member types 151
        panels 161
        project management 163
    enabling 345
    project and task duration 166
    tailoring projection definition 186
    user exits 168
    VX/VXA processing 188
LIFE-CYCLE member type 337
LIFE-CYCLE-OBJECT-TYPE member
    type 338
LINE-COMMAND 354
list panel
    definition 62
Local exit routines 189

M

macros 195
    discontinued 401
Mandatory property 12
MDC_CURSOR_COLUMN 184
MDC_CURSOR_ROW 184
MDC_LINE_NUM 184
MDC_LV 184
MDC_LV_LEN 184
MDC_LV_PROT 184
MDC_LV_SP 184
MDG_ABLAGE 157
MDG_ACTBEGIN 167
MDG_ACTDURATION 168
MDG_ACTEND 167
MDG_ADMIN_UDS 170
MDG_AKIN 160
MDG_AKOUT 160
MDG_AKTIV 152
MDG_APROT 149, 173
MDG_ATTSEP 111
MDG_AUFTRAG 153
MDG_AUPD_AMEND 117
MDG_AUPD_AMEND_EXCLUDE(N) 117

MDG_BLASTR 112
MDG_CHKMODEL 168, 185
MDG_COMMAND 157
MDG_COMMAND_LINE_CHAR 125
MDG_COMSTATE 158
MDG_CXEXT 149, 173
MDG_DELSTR 112
MDG_DKHELP 156
MDG_DKLAYOUT 156
MDG_DOKINC 130
MDG_DOKUMENT 154
MDG_EMPLASS 168, 185
MDG_ERGTYP 152
MDG_ESTDURATION 168
MDG_FILEXT(n) 149, 173
MDG_GEN_PANEL_EXIT 125, 163, 184
MDG_LCOT_INPUTS 165
MDG_LDISP 156
MDG_LDISPLEN 161
MDG_LDISPOFF 161
MDG_LINE_COMMAND_CHAR 126
MDG_LINE_PROTECTION_CHAR 116
MDG_LINE_PROTECTION_CODE 117
MDG_LOGOFF 126
MDG_MATRIX_SIZE_BATCH 128
MDG_MATRIX_SIZE_ONLINE 127
MDG_MIXED1 110
MDG_MIXED2 110
MDG_MMR_CX_CMD_LINE 50
MDG_MMR_CX_CMD_LINE(N) 123
MDG_MMR_CX_CMD_TYPE 50
MDG_MMR_CX_CMD_TYPE(N) 124
MDG_MMR_CX_HELP_TYPE 78
MDG_MMR_SET_AUTOSKIP 118
MDG_MMR_SET_BUFFER_LIMIT 118
MDG_MMR_SET_COMMAND_LINE 121
MDG_MMR_SET_LINE_COMMAND 120
MDG_MMR_SET_LINEAR_RETENTION 119
MDG_MMR_SET_LOOKASIDE_RETENTION 119
MDG_MMR_SET_OUTPUT_LINES 121
MDG_MMR_SET_PANEL_LIMITS 122
MDG_MMR_SET_UPDATE_OUTPUT 120
MDG_NAM_ENFORCE 138
MDG_NAM_NEW 139
MDG_NAM_OLD 136
MDG_NAM_OLD_MEM(N) 136
MDG_NAM_STD_ABBREV(N) 137
MDG_NAM_STD_NAME(N) 137
MDG_NAMEOL 133
MDG_NAMEON 133
MDG_NAMEXT1 149, 173
MDG_NAMEXT2 149, 173
MDG_NAMJOK 134
MDG_NAMNUM 134
MDG_NAMOPT 135
MDG_NAMSOL 134
MDG_NAMTST 139
MDG_NAMVAR 135
MDG_NOTE 154
MDG_PACTEXT 149, 173
MDG_PHAK 159
MDG_PHASE 152
MDG_PJLCASS 168, 185
MDG_PJVM 159
MDG_PLANBEGIN 166
MDG_PLANEND 167
MDG_PRJVIEW 153
MDG_PROJ_HIST_SWITCH 164
MDG_PROJDEF 168, 185
MDG_PROJECT_HISTORY 163
MDG_PROJECTVIEW_RUCOUNTS 166
MDG_PROJEKT 153
MDG_PROJSEL 168, 185
MDG_PROUTEXT 149, 173
MDG_PVIEW 165
MDG_RELABEL_BYPASS 169
MDG_RETAIN(n) 140
MDG_RETAIN_ALL 140
MDG_RETAIN_EX 140
MDG_RETAIN_MFR 140
MDG_RETAIN_MFS 140
MDG_SDISP 155
MDG_SDISPLEN 162
MDG_SDISPOFF 162
MDG_SECDEL 109
MDG_SKSTR2 113–114
MDG_SKSTR3 114
MDG_SKSTR4 114
MDG_SKSTR5 115
MDG_SKSTR6 115
MDG_SKSTR7 113, 115
MDG_STADEL 109
MDG_STEXT 149, 173
MDG_STINC 126
MDG_STMAX 127
MDG_STSEP 127
MDG_SYMOFF 113
MDG_TABLE_FIELD_CHAR 125
MDG_TASKSEL 168, 185
MDG_TOOL 155
MDG_TYPE 155
MDG_TYPLEN 163
MDG_TYPOFF 162
MDG_UPD_CLEANUPEXT 149, 173
MDG_UPDEXT 149, 173
MDG_UPDHEAD 112
MDG_UPDLOW 110
MDG_UPDRETEXT 149, 173
MDG_USER_AREA_1 128
MDG_USER_AREA_2 128
MDG_VMODELL 151
MDG_VMPH 159
MDG_VMUSER 158
MDG_VXEXT 168, 185
MDG_WBDA_ITEM_CHECK 141
MDG_WBDA_ITEM_PREF_NEW 143
MDG_WBDA_ITEM_PREF_OLD 142

MDG_WBDA_ITEM_REPLACE 141
MDG_WBDA_ITEM_SUFF_NEW 144
MDG_WBDA_ITEM_SUFF_OLD 143
MDG_WBDA_NAMING_EXIT 148
MDG_WBDA_RHSPRE 144
MDG_WBDA_SWITCH_PRSU_IT 142
MDG_WBDA_TABLE_PRSU(N) 147
MDG_WBDA_TABLE_TYPE(N) 145
member 17
    check-needed 39
    check-ok 39
    protection 173
    removing 41
    type 17
    validate 39
MEMBER-TYPE 286
    syntax 303
MEMBER-TYPE-GROUP 306
    syntax 309
menu definition 48
Meta-data 38
    interrogation of 43
MPR_EA60_DBODY 131
MPR_EA60_DECOMPOSE 132
MPR_EA60_HEADING 131
MPR_EA60_INDEX 132
MTHELP 362

N

naming convention 21
    customizing 133, 174
    generation 91
    global exit routines 174
    local exit routines 189

O

output panel definition 59

P

Panel Display
    tailoring 184
panel interface 45
    customization 123
    tailoring return to 193
    types 45
Panel Limits
    enforcing 122
Panel Processing
    tailoring 182
PHASE 343
PHASE member type 338
prerequisite 338
Project and task duration customization 165
PROJECT member type 337
PROJECT-VIEW member type 339
PROJVIEW 366
Property
    cardinality 11
    controlled 13
    dummies 14
    duplicates 15
    mandatory 12
    recursive 16
PX command 106

R

Recursive property 16
Relationship 10
    EA 17
    ER 17
    member type 17
    type 8
        class 43
        EA 17
        ER 17
        properties 10
RELATIONSHIP-CLASS member type 310
    syntax 311
RELATIONSHIP-GROUP member type 311
    syntax 313
RELATIONSHIP-TYPE member type 314
    syntax 332
Removing members 41
Repository information model 5
    implementing
        checklist of steps 36
        example 26
Return from Buffers customization 150
RIM
    see Repository information model
RULE010 90
RULE020 90
RULE030 90
RULE040 90
RULE050 90
RULE070 91
RULE080 91
RULE100 91
RULE110 91
RULE120 91
RULE130 91

S

SEXEC member type 335
Simulation 345
Standard abbreviation table 137
Subactivity 338

T

Tailoring the environment 105
TASK member type 338
Tool Usage Model 338

U

UDR clauses 91
UDS-TABLE member 89

User exits 171

V

validation of members 39
VX command 345
VX processing
    tailoring 188
VXA command 345
VXC command 345
VXP command 346

W

Workbench Design Area
    customization 141