



RE / m

*Reverse Engineering
for M Applications*

RE/m 2.0 Reference Manual

Contents

Introduction	5
Operation Guide	7
System Functions	13
Reverse Engineering	17
Parser Definitions	31
Process Catalog	45
Data Catalog	59
Process Selection	65
Process Structure Diagrams	75
Global Structure Diagrams	81
Process Usage Matrices	87
Process/Data Matrices	93
Record Content Diagrams	101
Appendix A - Installation and Configuration	107

Notices

Copyright 1990-1996 George James Software Limited.

No part of this publication may be reproduced, transmitted, stored in a retrieval system, nor translated into any human or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of the copyright owner, George James Software Limited, Shepperton Marina, Middlesex, TW17 8NS, England.

Information contained in this publication is subject to change without notice and does not represent a commitment on the part of George James Software Limited.

Any copyrighted software accompanying this publication is licensed to you only for use in strict accordance with the Software License Agreement accompanying the software. Please read the license agreement carefully before commencing use of the software.

RE/data, RE/m, RE/parser and VC/m are trademarks of George James Software Limited. All other brand and product names are or may be trademarks of, and are used to identify products and services of, their respective owners.

Order code 1001.

Product Support

Support is available to all users of George James Software Products who have a current Software Maintenance Agreement. Support and assistance can be obtained from the following sources:

Telephone	+44-1932-252568
Fax	+44-1932-254816
E-mail	support@georgejames.com
Web page	www.georgejames.com/support

Revision	File Reference	Date	Author	Notes
0	t:\gj\doc\remref20.0	30 July 1999	George James	
1	\\Wiz\D\Lib\Manuals\RE m\remref20.1.doc	28 Sept 2000	Gail Treves Brown	reformatting
2	\\Wiz\D\Lib\Manuals\RE m\remref20.2.doc	18 June 2002	Gail Treves Brown	latest logo
3	\\Wiz\D\Lib\Manuals\RE m\remref20.2.doc	5 December 2002	Michelle Stevenson	Minor amendments

CHAPTER 1

Introduction

Contents

Introduction 6

RE/m is a repository based reverse engineering tool for M applications. It generates documentation of M software that can be used to aid the support, maintenance and enhancement of that software. RE/m consists of three major components:

- o A sophisticated and flexible code analyzer. The code analyzer reads the source code for M programs and derives data usage and structural information from them. This information is stored in a repository database as process and data objects.
- o Cataloging facilities provide a way of grouping and documenting the contents of the repository in a manageable form. Both process and data objects can be cataloged at five different levels:

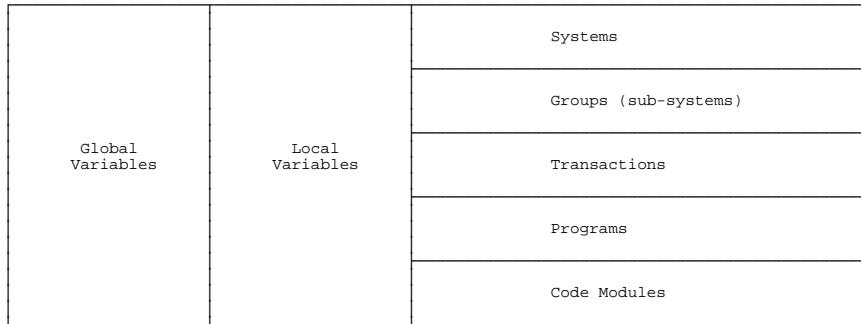


Figure 1.1 The RE/m repository

The top level catalogs provide a business application based view of the physical programs and files that make up a system. This enables systems analysts with limited technical knowledge to be able to interrogate and interpret the repository.

- o Finally, a set of analysis tools provide a powerful and flexible method of interpreting the contents of the repository to aid the analyst and programmer in the tasks of enhancing, maintaining and supporting an application.

Chapter 4 of this reference manual describes how to use the code analyzer to load the repository. Chapter 5 explains how to customize the parser to suit the environment in which it is being used. Chapters 6 and 7 describe the cataloging facilities provided by RE/m and provide guidelines on their use. The remaining chapters of the user manual cover the analysis tools for interpretation of the repository.

Appendix A contains the installation procedure for the first time installation of RE/m. It also contains instructions for the configuration of RE/m to work with different terminals, PC's and printers.

Operation Guide

Contents

Getting Started	8
Menu Operation	8
Input Fields	9
Selection Windows	10
Input of Directory Specifications	12

This chapter describes how to operate the system and explains the basic user interface.

Getting Started

RE/m will run on any terminal that conforms to ANSI standard 3.64 (eg VT220, VT420, and most PC terminal emulators). If your terminal is non-standard or you have problems using the system consult your system manager.

To use RE/m, log in to the area where RE/m is installed and enter the following command:

```
>d ^re
```

or

```
>D ^RE
```

You will be presented with following screen and the main menu:

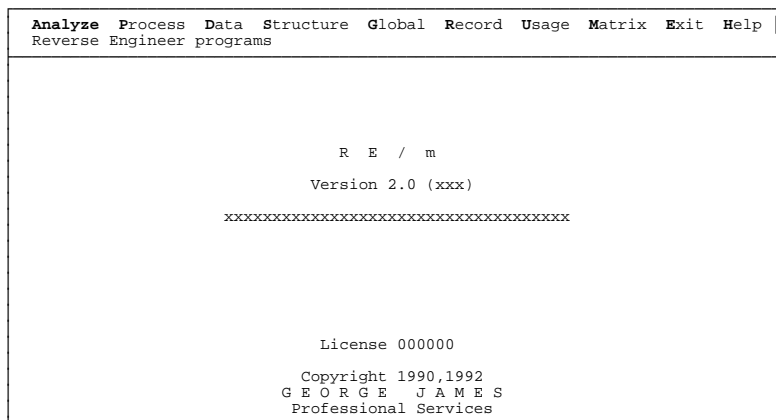


Figure 2.1 RE/m front screen

Menu Operation

All the menus within the system use a menu bar style that conforms to de-facto industry standards. Each menu consists of a series of options arranged horizontally on the first line of the screen, If there are more options than will fit on the screen then the menu will scroll horizontally as the cursor is moved across.

The default option is highlighted in inverse. The cursor keys (left-arrow and right-arrow) can be used to move the highlight from one option to the next. The second line of the screen contains a longer description of each menu option when it is highlighted. If the last option on the menu is highlighted and the right-arrow key is pressed the first option will be highlighted and vice versa.

There are three ways to select an option from the menu bar:

- 1 Position the highlight over the desired option and press the <return> key.
- 2 Position the highlight over the desired option and press the down-arrow key.
- 3 Press the key corresponding to the character in the menu option that is underlined or highlighted.

Every menu bar has two standard options, Exit and Help. If exit is selected then you will be taken back to the previous menu from which you came. If help is selected then you will obtain a page of help text that describes the functions that are available to you.

The up-arrow key has the same effect as selecting the exit option.

Input Fields

All input fields are represented by a prompt, normally to the left of the field, and a reply area enclosed by square brackets. For example:

```
Program [      ]
```

If the field has a default then this will be displayed in the reply area. To accept the default press the <return> key.

To go back to a previous field press the up-arrow key. NB in some cases the up-arrow key may not work (for example, under VMS if line editing is enabled or escape processing disabled). If it does not work then backslash (followed by <return>) may be used and will have the same effect.

To delete the contents of a field enter a single space followed by <return>. If the field is mandatory you will not be allowed to delete it.

On the rare occasions when you need to enter a field consisting of a single space, enter two spaces. RE/m will treat this as a single space.

Selection Windows

At most input fields where one or more items can be chosen from the repository a selection window can be invoked to browse through the contents of the repository and to make your selection.

To invoke a selection window enter one of the following keys, depending upon the type of terminal that you are using:

Terminal Type	Keystroke
PC Console	F1, or ?
VT220, VT320, VT420	HELP key, or PF1, or ?

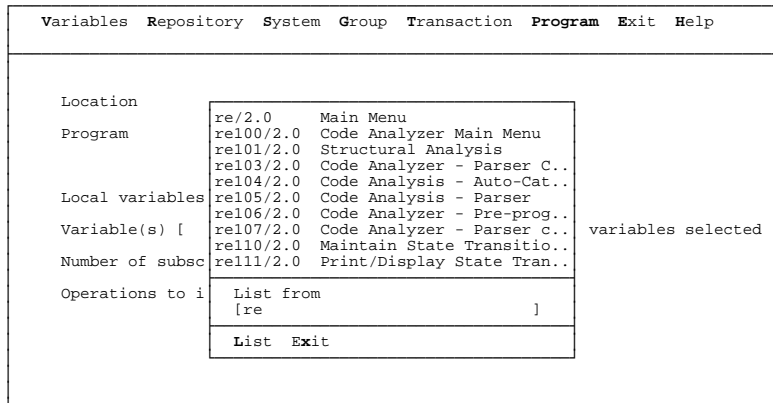


Figure 2.2 Single item selection window

There are two types of selection window. The single selection window (figure 2.2) allows you to browse through the contents of the repository and to select one item using either the space bar or by pressing <return>.

The second type of window (figure 2.3) is used when a multiple selection is possible (ie when selecting routines or variables). In this case the window contains two sections. The left hand section displays the contents of the repository. The right hand section displays the items that have been selected.

You can move between the two sections by using the left and right cursor keys. If you select an item in the left hand window it will be added to the list in the right hand window. If you selected an item in the right hand window it will be removed from the list.

There is also a Select prompt, which can be reached by typing S. This is a conventional selection prompt that enables items to be selected and/or de-selected from the right hand list using wildcards and ranges.

To display the contents of the repository window from any arbitrary start point, type L. This will take you to the *List from* prompt. At this prompt you can then enter the start point.

In a multiple selection window once you have made your selection type X to return to the original screen.

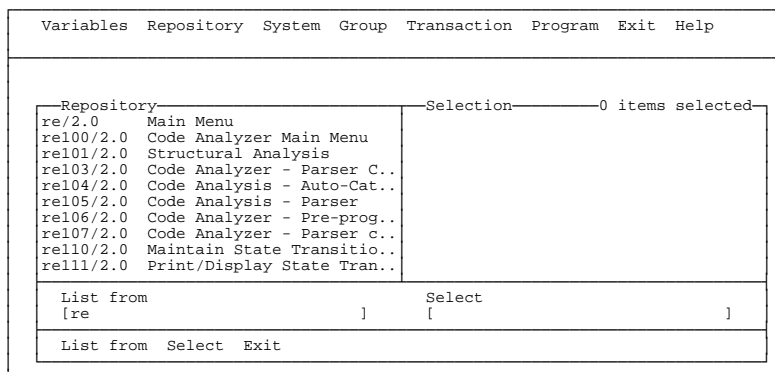


Figure 2.3 Multiple item selection window

The following table summarizes the keystrokes that can be used to navigate within a selection window:

Summary of Keystrokes

Cursor Up	Scrolls up the list by one line
Cursor Down	Scrolls down the list by one line
Cursor Left	Moves from the selection window to the repository window
Cursor Right	Moves from the repository window to the selection window
Page Up	Scrolls up the list by one page
Page Down	Scrolls down the list by one page
X	Returns to the original screen
S	Goes to the Select prompt
L	Goes to the List from prompt

Input of Directory Specifications

When a directory specification is prompted then normally this should be input using the appropriate format for the host operating system. There is one exception to this. If RE/m is running under MS-DOS then the directory specification should be entered using a forward slash character to delimit sub-directories rather than a backslash character. This is because the backslash character is used as an alternative escape key and, if it occurs in any input string, is interpreted by RE/m in the same way as the Cursor Up key.

For example:

```
C:\DTW\ROUTINES.RSA
```

Should be entered as:

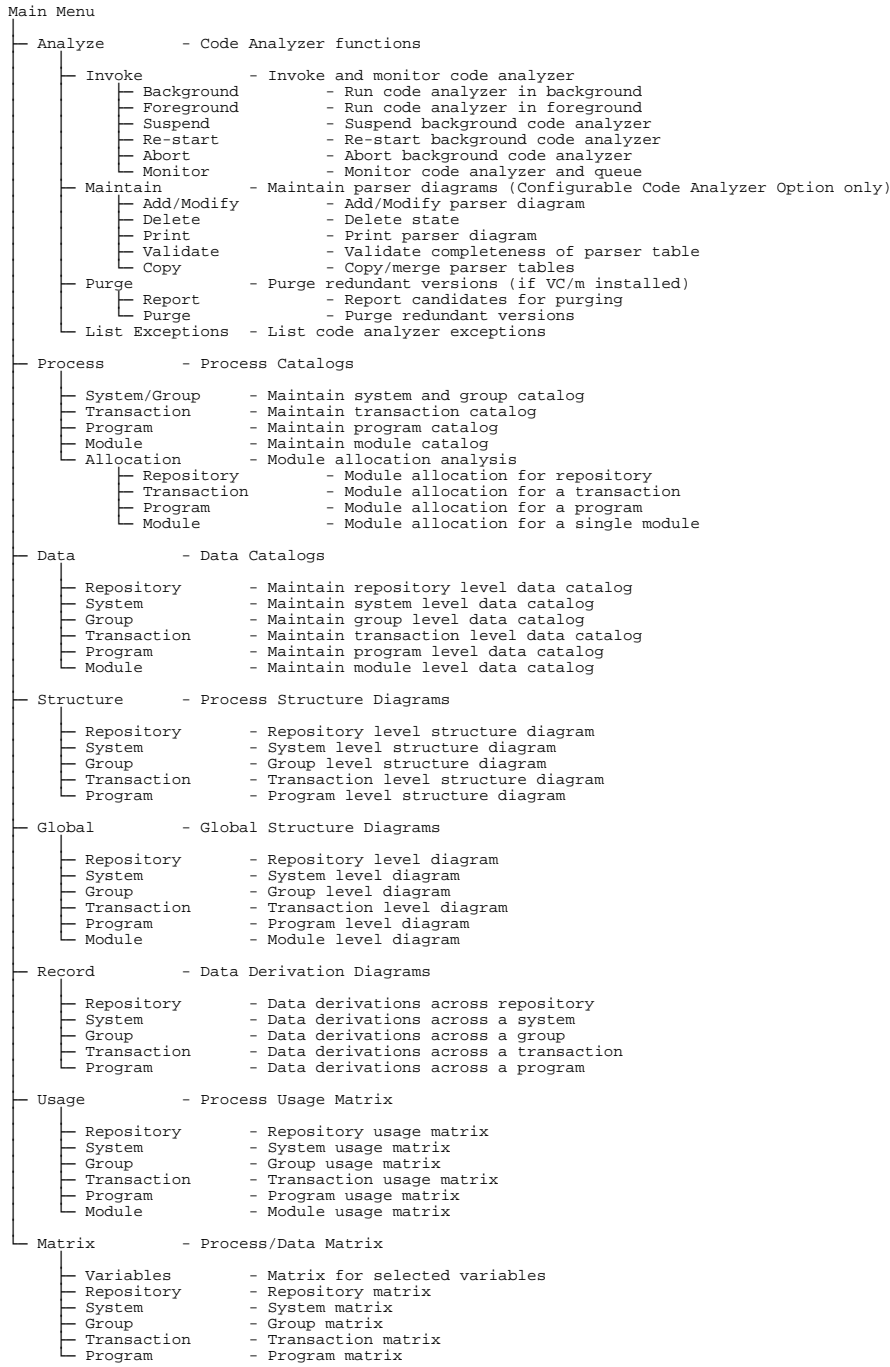
```
C:/DTW/ROUTINES.RSA
```

System Functions

Contents

System Functions 14

The following diagram provides a quick reference guide to where each function in the system can be found:



Reverse Engineering

Contents

Repository Updates	18
Code Analyzer	19
Invoking The Code Analyzer	20
Exception Report	24
Purging Redundant Versions	24
The Repository	25
Indirection	27
Automatic Invocation Of The Code Analyzer	29

The RE/m code analyzer parses M source code using a pre-selected parser table. There are three parser tables shipped with RE/m:

- o **RE2.0** This is the default parser table. It analyzes code written using any implementation of M and updates the repository with the results of that analysis.
- o **REansi90** The REansi90 parser analyzes M code written to the 1990 ANSI standard. It will report as exceptions any code that does not conform syntactically to this standard. The result of the analysis is used to update the repository. This parser table should be used if you are interested in ensuring that your code conforms to the 1990 standard.
- o **Vanilla90** The Vanilla90 parser table permits ANSI 1990 syntax only and will report any other syntax as exceptions. This parser table does not update the repository. It is provided as a starting point for users that wish to build custom parser tables that perform tasks such as conversion of applications from one implementation of M to another, or for identification of specific syntactic elements such as string literals.

Repository Updates

In general, if a parser table has a name that begins with RE it will update the RE/m repository with information about the programs that have been analyzed. Each program read by the code analyzer is first broken down into discrete code modules. Each code module represents a piece of code with a contiguous structure. Each module is then further analyzed to determine its relationship with other modules (ie what it calls and what it is called by) and to determine how it uses data (both local and global variables). The code analyzer is driven by a parser table which defines the syntax of the code being analyzed and the way it is interpreted as information to load into the repository. If the Configurable Code Analyzer option is purchased the parser diagrams can be user modified to adopt the code analyzer to the environment in which it is used. In particular the parser diagrams can be modified to cater for:

- o Implementation specific M language extensions
- o Meta-language syntax that is translated by a pre-processor
- o Validation and enforcement of coding standards
- o Validation against data dictionaries
- o Automated code conversion

It is recommended that the whole of chapter 4 is read before code analysis is performed on a large number of programs.

Code Analyzer

To use the code analyzer a sequential file must first be prepared containing the M programs to be analyzed. This file should be in DSM Routine Save format and can contain one or more M programs.

The code analyzer can be invoked to run either in foreground or as a background process. Normally it should be run in background so that it does not tie up a terminal while it is running. The Monitor menu contains an option that enables the progress of the code analyzer to be observed. It also contains options to suspend, re-start and abort the code analyzer.

Only one code analyzer can be run at any one time, either in foreground or in background. If several Routine Save files are submitted for analysis then they will be held in a queue and will be processed one at a time in the order in which they were submitted.

While the code analyzer is working, the monitor progress display indicates the current program being analyzed and statistics for the programs analyzed so far (Figure 4.1).

Analyze Process Data Structure Global Record Usage Matrix Exit Help					
Reverse Engineer programs					
Invoke Maintain Purge List exceptions Exit Help					
Invoke and monitor code analyzer					
Background Foreground Suspend Re-start Abort Monitor Exit Help					
Monitor background code analyzer					
Active parser table RE2.0					
File	suppl10.rsa	Total routines	13	Lines	333
Routine	HAPQ102	Current line	50	Rate	321
				Statements	390
				Exceptions	29
		Total		State-	Except-
		Rtns	Lines	ments	ions
46	Pending				
	cust.rsa				
45	Analyzing				
	suppl10.rsa				
44	Complete				
	dual:system.rsa	309	19928	37266	680 481
43	Complete				
	c:\dt4\rtns.rsa	64	1671	4023	747 2
42	Complete				
	manager.rsa	38	2269	6934	561 1
41	Complete				
	qctest07.rsa	2	173	410	683 1
40	Complete				
	qctest06.rsa	47	5050	3385	329 0
39	Complete				
	qctest05.rsa	2	362	439	712 0
38	Complete				
	qctest04.rsa	1	22	17	340 0
37	Complete				
	qctest03.rsa	3	230	275	375 0
36	Complete				
	qctest02.rsa	3	192	275	367 0
35	Complete				
	qctest01.rsa	6	2356	2543	241 72
34	Complete				
	temp2.rsa	94	15764	16966	286 1
33	Complete				
	templ.rsa	21	4504	4815	330 2

Figure 4.1 The code analyzer progress display

Invoking the Code Analyzer

The code analyzer can be invoked using either the Foreground menu option or the Background menu option. If the Foreground option is selected then the code analyzer will be invoked directly in foreground and will display its progress on the screen. If the Background option is selected then the Routine Save file to be analyzed will be submitted to a queue. If the code analyzer is not already running in background then it will be automatically started and any files in the queue will be analyzed.

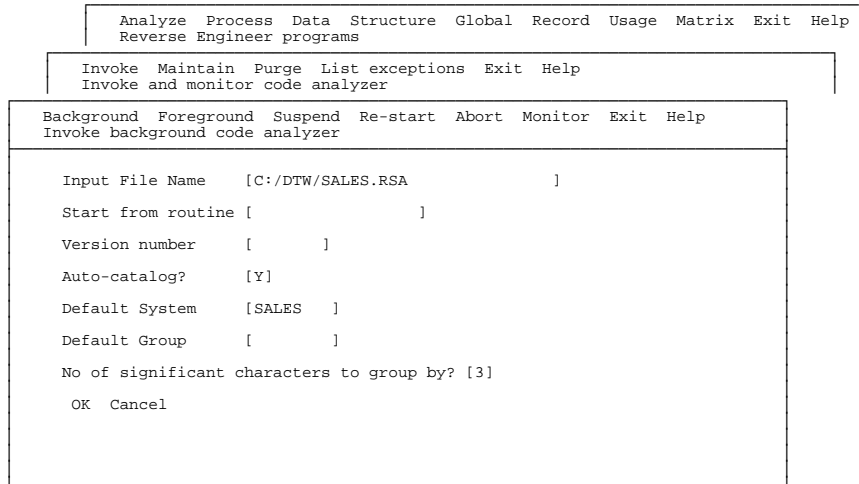


Figure 4.2 Code Analysis

There are a number of inputs that are required when the code analyzer is invoked. They should be entered as follows:

o **Input File Name**

Enter the file name of the file containing the DSM-11 format Routine Save of the M code to be analyzed. If the file is not in the current default directory then the full file specification or path should be entered to identify the file uniquely.

If you are using an MS-DOS environment then please note that file specifications should be entered using a forward slash rather than a backslash. For example:

C:/DTW/ROUTINES.RSA

o **Start from routine**

If you have a very large Routine Save file and you do not wish to analyze all the routines in the file then it is possible to specify the name of the routine at which the code analyzer should start. All routines before this will be skipped.

If you wish to analyze all routines in the input file then leave this field blank.

o **Version number**

If you wish to analyze more than one version of the same routine and to store the results of each version in the repository then each routine can be labeled with a version number.

There are three ways of labeling each routine with a version number. If a version number is entered at this prompt then each routine within the routine save file will be given this version number. The version number should be entered in the format XXX.999 where XXX is the major part of the version number and 999 is the minor part of the version number.

The second way of labeling each routine with a version number is to use a modified form of Routine Save file that contains the version number as a suffix to the routine name. The format of the routine name line in the routine save file should be as follows:

routine name/XXX.999

where *routine name* is the name of the routine that follows
 XXX is the major part of the version number
 999 is the minor part of the version number

The third way of labeling each routine with a version number is to use a transfer file created by VC/m (our Version Control and Configuration Management System). If a VC/m transfer file is to be analyzed then at the Input file name prompt enter the name of the transfer file with an extension of \$. This is the control file for the VC/m transfer file set and contains version number and control information about the routines contained in the routine transfer file.

o **Auto-Catalog**

If the set of M routines being analyzed have been developed using a rigid program naming convention then it is possible to create the transaction level of the repository automatically during code analysis.

In order to be able to do this the programs must have been developed so that each transaction (or task, function, job, business event or logical process, depending upon the terminology that you are familiar with) can be identified as a set of routines by the first few characters of the program name. The following example illustrates this:

Program	Purpose of program	Belongs to transaction
SLA1	Account Maintenance Controller	Account Maintenance
SLA2	Account Maintenance Add	Account Maintenance
SLA3	Account Maintenance Delete	Account Maintenance
SLA4	Account Maintenance Revise	Account Maintenance
SLB1	Account Enquiry Controller	Account Enquiry
SLB2	Account Enquiry Summary	Account Enquiry
SLB3	Account Enquiry Detail	Account Enquiry
ACMNT	Account Maintenance Controller	Account Maintenance
ACCADD	Account Maintenance Add	Account Maintenance
AMNTDEL	Account Maintenance Delete	Account Maintenance
REVACC	Account Maintenance Revise	Account Maintenance
ACCENQ	Account Enquiry Controller	Account Enquiry
ACCENQS	Account Enquiry Summary	Account Enquiry
ACDETAIL	Account Enquiry Detail	Account Enquiry

Figure 4.3 Auto-cataloging

If the set of programs has been developed using a rigid naming convention such as the first set then each transaction can be defined automatically as all programs that match on the first three characters of their name. If the set of programs being analyzed uses an informal naming convention, as in the second example above, then auto-cataloging cannot be used to automatically create transactions in the repository.

If Auto-cataloging is selected then the following information must be supplied at the start of code analysis:

o **Default System**

The name of the system to which the programs being analyzed belong. All transactions created automatically will be linked to this system. If a new system is being cataloged then it will be necessary to set up a system entry in the repository before the code analyzer is run. A system entry can be set up by selecting the Process option on the main menu and the System/Group option on the next menu.

- o **Default Group**

If the programs being analyzed all belong to the same group within the selected system then the name of that group can be entered here. If the programs belong to more than one group, or the system is not broken up into groups then leave this field blank. The transaction entries that are created automatically can be modified to alter the system or group at any time after the code analysis is complete.

- o **Number of significant characters to group by**

This field determines the number of characters of the program name that identify a transaction. For each set of programs that comprise a transaction the code analyzer will automatically create a transaction entry with a name comprising of the common characters of the program name. This transaction will be linked to the system and (optionally) group specified at the start of the analysis session. All the code modules within each program in the set will be automatically linked to the transaction entry. If a code module has previously been linked to a different transaction then it will not be automatically linked to the transaction created automatically.

If a matching transaction entry already exists then all code modules within the program will be linked to that transaction. If other code modules have already been linked to the transaction they will not be deleted.

Note that the analysis of a large number of programs can take a considerable length of time. If the analysis is being performed in foreground then it can be stopped, in an emergency, using Ctrl-C however this will leave the repository in an unpredictable state. If the code analyzer is being run in background then it can be stopped using either the Suspend or the Abort option on the monitor menu. In this case the analyzer will complete the analysis of the current program and then stop leaving the repository in a consistent state. For this reason it is recommended that the code analyzer is always run in background.

The code analyzer can create a database of a significantly large size. At least 10 Mbytes should be allowed for each 1,000 programs being analyzed. Before running the code analyzer remember to check that there is sufficient free disk space and, if necessary, that the database has been expanded to allow sufficient room.

Exception Report

Any code encountered that is not understood by the code analyzer is treated as an exception.

Exceptions can be examined using the List exceptions menu option at any time. A report similar to that in figure 4.4 can be displayed on the screen or printed showing each line that contains an exception and an indication of where on the line the exception occurred.

In parenthesis following each error message is the name of the parser diagram state that gave rise to the exception. If the exception is the result of parser table customization and was not intended then this can be used as an aid to identifying which parser diagram is responsible for the problem.

```

Analyze Process Data Structure Global Record Usage Matrix Exit Help
Reverse Engineer programs

Invoke Maintain Purge List exceptions Exit Help
Invoke and monitor code analyzer

Show Print Maintain Exit Help
Display analyzer exceptions on the screen

Line: +48^sla3          Syntax Error (SET4)
a100      s x=1:1:10 s arr(x)^=sl(sla1,sla2,x)
          ^

Line: +10^sla4          Expected " at end of line
          i sla3'?1ul.e s %m="Invalid Input d msg q
          ^

Line: +15^sla4          Syntax Error (D03)
a320      i arr(x)=" d a400,x'=1,a500
          ^

```

Figure 4.4 Code Analysis Exception Report

Purging Redundant Versions

The Purge option within the Analyze menu purges redundant and old versions of routines. It is only applicable if VC/m is installed and is used to control the contents of the RE/m repository.

Any version of a routine that exists in the repository will be deleted automatically using this option unless it has a status of *active* in the VC/m configuration management database.

A purge report can be produced to list all routines that will be deleted from the repository if the purge is actually run.

If this feature is to be used then the VC/m routing should be set up so that any version of a routine which is of interest in the RE/m repository should have a status of *active* in the location that maps onto the repository. When it is no longer of interest, perhaps because it has been superseded by a later version, then its status should be changed to *inactive* within the repository. The RE/m purge function will then be able to physically delete it from the repository.

NB If a program is actually deleted using VC/m then when a transfer file containing the deletion is analyzed the program will be removed from the repository at this time. In contrast, the purge facility is designed to remove old superseded versions of a routines on a periodic basis.

The Repository

When the code analyzer is run, using either the RE2.0 or REansi90 parser tables, it creates entries in the repository at program and module level, for each program in the input file. They are created as follows:

o **Program Entries**

For each program, an entry is created in the program catalog. A program description is derived from the first comment encountered within the program.

If a program has previously been Reverse Engineered the information in the repository will be updated to reflect the difference between the old version of the program and the new. Any manually entered documentation about the program will be automatically retained unless the corresponding module no longer exists.

o **Module Entries**

Each program is analyzed to identify code modules. A record is created in the repository for each code module. It is normally identified by the name of the first label within the module. A module description is derived from the first comment encountered within the module.

o Local and Global Variables

The usage of each local and global variable within a program is recorded in the repository along with the mode of use (Set, Kill, Lock etc). References to variables using indirection (eg $S @X=1$) will be stored in the repository as they appear (eg $@X$). Subscripted variables are *regularized* so that each subscript level is replaced by a hyphen. This is so that the same variable reference using different subscripts would be identified as being synonymous.

In figure 4.5 examples 1, 2 and 3 show global accesses to the same global node, but using different variables as subscripts. In the repository these would all be recorded as having the same, regularized, global reference. Where a subscripted variable is referenced using a constant, the constant does not get regularized. Example 4, in figure 4.5, shows how a global reference with constant keys would be recorded.

Example	Variable Reference	Repository contents
1	$^sl(sla1,sla2)$	$^sl(-,-)$
2	$^sl(x,y)$	$^sl(-,-)$
3	$^sl(x,arr(y)+2)$	$^sl(-,-)$
4	$^pp("A",ptno,10)$	$^pp(A,-,10)$

Figure 4.5 Regularization of subscripted variables

o Data Derivations

The code analyzer identifies and records instances of data derivations. This indicates how each variable is derived from other variables, constants or literals. This information is primarily used to determine the field structure within global nodes and hence produce record layouts.

o Variable Descriptions

The code analyzer uses two techniques to identify descriptions for local variables. Firstly if a literal is output using a *write* command and this is followed immediately by a *read* or *write* of a variable, then the literal is associated with that variable as a first-cut description.

Secondly if a variable is *set* to a literal string then that literal is recorded in the repository as an instance of the variable. In these cases the literal will be enclosed in quotes to distinguish it from descriptive literals.

Indirection

Any variable that is used as an indirect argument to a DO, GOTO or JOB command can be declared to RE/m as an indirect variable. Whenever the variable is set to a literal that is a label or routine reference the code analyzer will automatically interpret this as a reference to the named label or program and establish a cross reference relationship in the repository. These relationships will then be treated in exactly the same way as DOs and GOTOs in structure diagrams and matrices.

To declare a variable as an indirect variable it is necessary to use the RE/m setup program (D ^RESETUP). On the main menu select the *indirection* option and within the indirection sub menu select the *Add/Modify* option and enter the name of the variable.

```

Device Types  Terminals  Printers  Indirection  Exit  Help
Maintain Indirect Variables

Add/Modify  Delete  Show  Print  Exit  Help
Add/Modify indirect variables

Variable          [COLUMN          ]
Mode of operation [D]  DO
Comment           [Column heading sub-routine      ]
Document Ref      [Programming Standards Section 2.4  ]

```

Figure 4.6 Identifying indirect variables

o Variable

The name of the variable that is used as an argument to indirection should be entered. Subscripted variables may be entered if required but note that they should be entered in their regular form, for example, SUBR(- ,SC, -)

o Mode of Operation

Enter the mode of operation in which the variable is used. It can be defined as an argument to a DO command, a GOTO command or a JOB command. This information will be used to determine the kind of relationship shown on the structure diagrams and in the usage matrices.

o **Comment**

This field is a free text comment enabling the purpose of the variable to be documented.

o **Document Ref**

This field enables a cross-reference to be made to an external piece of documentation, such as programming standards or a technical specification.

The code analyzer will only create indirect relationships for variables that have been declared. It is therefore worthwhile declaring any indirect variables that are required before large numbers of routines are analyzed.

The treatment of indirect variable is in two parts. The static part is to create a relationship between the module where the variable is set and the module that is referenced by the value of that variable. The dynamic part is the creation of a relationship between any module that makes an indirect reference to the variable and the variable itself.

The result of this is that structure diagrams will contain a direct link between the module where the indirect variable is set up and the module invoked. They will also contain an indirect link from the module where the indirect variable is used to the variable itself. There will not however be a link between the indirect variable at the place it is invoked and the module that is invoked.

Automatic Invocation of Code Analyzer

The code analyzer can be invoked automatically using one of the following entry points:

```
DO ^re310(file,autocat,autosys,autogrp,automat,version,from)
JOB ^re310(file,autocat,autosys,autogrp,automat,version,from)
DO queue^re310(file,autocat,autosys,autogrp,automat,version,from)
```

The entry point ^re310 runs the code analyzer directly. The entry point queue^re310 submits a request to analyze to a queue and starts the code analyzer in background unless it is already running.

The arguments to these function calls are as follows:

file	=	The name of the file containing the routines to be analyzed. Mandatory.
autocat	=	Auto-cataloging? Y = Yes, N = No. Defaults to N.
autosys	=	Default system name. Mandatory if autocat=Y.
autogrp	=	Default group name. Optional.
automat	=	Number of characters to match by? Mandatory if autocat=Y.
version	=	Version number suffix to routine names. Optional.
from	=	Name of routine to start analyzing from. Optional.

Parser Definition

Contents

Parser Diagram Notation	34
Editing Parser Diagrams	39
Validation	43

Note: This chapter describes the facilities and procedures for customizing the code analyzer. The code analyzer is pre-configured to support ANSI standard M, so it is not necessary to customize the code analyzer for normal use. The parser diagram customization procedure is complex and should only be performed after suitable training.

The customizable code analyzer is available as an option. If the option has not been purchased then the facilities described in this chapter will not be available. The Customizable Code Analyzer can be purchased later as an add on option if it was not purchased initially.

The code analyzer is driven from a set of tables. Each parser table is represented in diagrammatic form as a set of parser diagrams.

Three different parser tables are supplied with RE/m. Each is configured to perform a different function:

o **RE2.0 - RE/m Version 2.0 Default Parser Table**

This parser table is the default table. It has the following features:

- o Full support for the 1990 ANSI standard of the M language.
- o Z command and function extensions are supported in a 'pass-all' mode. IE any Z command will be permitted by the parser. Z commands will not be reported as exceptions and no meaningful information will be derived from the arguments of these commands. It is recommended that the appropriate language extensions used by your M implementation be added to the parser diagram set, if they are used within the software being analyzed.
- o The ZA and ZD commands are explicitly interpreted as being equivalent to incremental locks.
- o The square bracket extended global syntax is interpreted as valid syntax and treated semantically as a qualifier to the name of a global.
- o The proposed standard syntax for mnemonic device handling is supported. For example w /c(x,y)

- o The proposed standard syntax for reverse \$order is supported.
- o The proposed standard syntax for SET \$EXTRACT is supported.
- o The proposed standard syntax for the PRINT command is supported.
- o The parser table includes calls to user sub-routines to populate the RE/m repository. Changes to this parser table can affect the extent and manner in which the repository is created.

o **REansi90 - ANSI 1990 Standard Parser Table with Repository Update**

This parser table permits only the syntax defined in the 1990 ANSI standard. It does not permit Z command and functions and will report them as exceptions even if they are syntactically valid extensions to the language.

This parser includes the calls to the user sub-routines that update the RE/m repository to populate it.

o **Vanilla90 - ANSI 1990 Standard Parser Table with Repository Update**

This parser table only permits syntax conforming to the ANSI 1990 standard for M.

It does not perform any updates to the repository.

In its unmodified form this parser table does nothing other than parsing M code and reporting anything that does not conform to the 1990 standard. It is supplied as a starting point for creation of custom parsers that can perform any required function from scanning for specific syntactic elements to automated code conversion.

Appendix B contains a printout of the main parser diagrams contained within this parser table.

RE/m provides the ability to modify these diagrams to customize the code analyzer to handle local conventions and implementation specific syntax.

It is recommended that the parser tables supplied with RE/m are not modified directly. Instead the table should be copied and changes applied to the copy. A copy/merge option is provided within the Maintain Parser Diagrams menu for this purpose.

It is further recommended that modifications be made to an empty parser table and then merged in to a copy of one of the supplied parser tables. This will enable you to re-apply your customization in a clean and quick way when you receive updated versions of the standard parser tables. Figure 5.1 illustrates how a sub-set of parser diagrams called *Standards* is merged into a copy of the Vanilla90 table called *Working*.

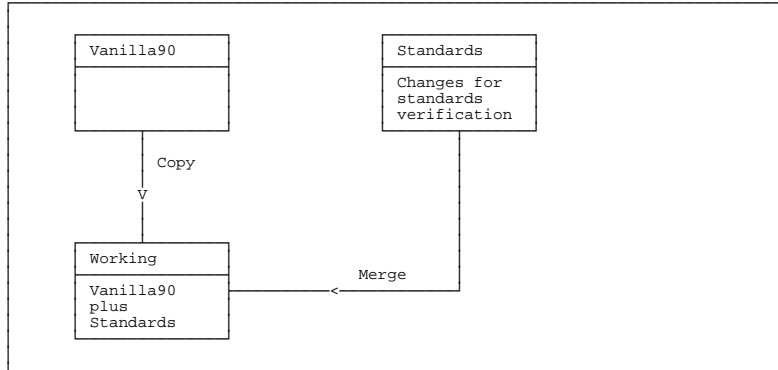


Figure 5.1 Creation of a custom parser table

Diagram Notation

Each diagram consists of a series of states linked by transitions. Each state represents a specific point in a piece of syntax. As the code analyzer reads the code in a program it uses the parser diagram as a map or guide that tells it what to expect next and what to do with it. Each transition represents an alternative valid path for the code analyzer.

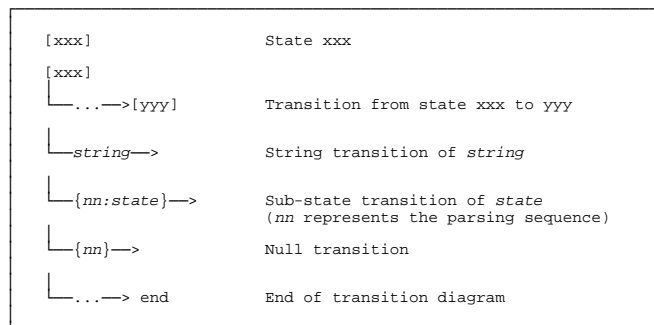


Figure 5.2 Parser Diagram Notation

Figure 5.2 summarizes the notation used within parser diagrams. At each state there are three alternative types of transition:

o String Transition

For each defined string transition, the code analyzer matches the following code with the permissible string transitions. If a match is found the code analyzer moves to the new state and processes the transition in accordance with the transition definition.

Figure 5.3 shows an example of the diagrammatic representation of string transitions. In this case, if the code analyzer is at state SET1 and the next string in the code under analysis is 'SET' then the code analyzer will move to state SET2. If the code analyzer encounters the string 'set' then it will also move to state SET2.

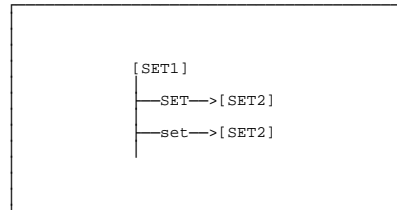


Figure 5.3 String Transitions

o Sub-state Transition

If no string match is found then each possible sub-state is processed in turn. Each sub-state consists of a complete parser diagram. If a sub-state is parsed successfully then the code analyzer moves to the new state following that sub-state and processes that. If a sub-state is not parsed successfully then the next sub-state in the sequence is parsed and so on until a valid sub-state is found. If none of the possible sub-states are valid then the code analyzer will stop analyzing the line, record it as an exception and continue with the next line.

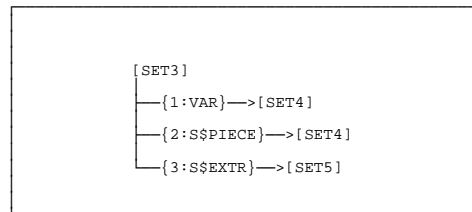


Figure 5.4 Sub-state Transitions

Each sub-state has a posit sequence number. Each posited sub-state is parsed in sequence until a sub-state is successfully parsed. When this occurs the code analyzer moves to the new state.

Figure 5.4 shows an example of the diagrammatic representation of sub-states. In this example, when the code analyzer is at state SET3 it will first attempt to parse the code through state VAR. If this is successful then the code analyzer will move to state SET4. If the code is not parsed successfully by VAR the code analyzer will attempt to parse it with S\$PIECE and then with S\$EXTR. If the code is not successfully parsed by any of the sub-states then the code analyzer will record an exception.

o **Null Transitions**

A null transition provides a way of moving from one state to another without parsing any source code at all.

Null transitions are specified in exactly the same way as sub-state transitions except that a sub-state is not defined. A null transition has a posit sequence number which enables it to be placed after other possible sub-states if necessary.

Figure 5.5 shows an example of a null transition. If neither of the posited sub-states are valid then the code analyzer will move from state SET4 to state ERROR.

The Vanilla90 parser table contains a sub-state called NULL which will also perform a null transition. Figure 5.6 shows how this can be used to produce the same result.

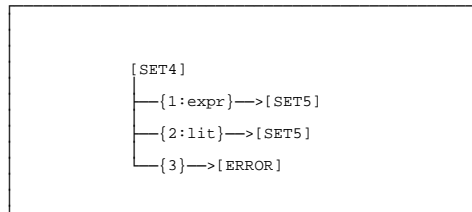


Figure 5.5 Null Transition

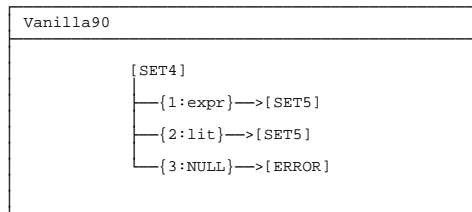


Figure 5.6 Null Transition (Alternative)

For each transition there are three ways the parser can interpret the code being parsed. They are:

o **Flag Setting**

One or more flags can be set to indicate that a particular state has been encountered. For example, when a transition is made to state Z a flag might be set to indicate that an M language extension has been encountered. Each flag is one character in length, therefore multiple flags can be set at any one state transition.

When a flag is set it is stored in a global node that can be examined by user sub-routines at other points in the parser table. Each flag is stored in the following global node:

```
^retmp($j, "F", lin, flag) = " "
```

where *lin* is the line number of the line being analyzed
flag is the *single character* flag that has been set

The flags are reset at the start of each routine that is analyzed.

NB the RE2.0 and REansi90 parser tables use the U flag to identify the end of a module. The U flag should not therefore be used unless it is for this purpose.

o **User Sub-Routine**

A user defined sub-routine can be invoked to perform custom processing, such as validating the existence of a variable in a data dictionary. Much of the RE/m repository is created using custom sub-routines, this enables the way that the repository is updated to be adopted to the user's own requirements if necessary.

The following local variables are available at execution time within a user sub-routine and can be referenced by the sub-routine:

arg	Line of source code currently being parsed.
x	Pointer to the position in arg of the character after the string or sub-state that has just been successfully parsed.
y	Result of parsing a state.
routine	Name of the routine currently being parsed.
lin	Line number within the current routine.
zcmd	Number of statements parsed in the current routine (In the parser tables that update the repository this variable represents the number of commands parsed in a single module).
out	Output buffer for sub-state currently being analyzed. If the parser is being used for code conversion then this can be modified by <i>appending</i> data to the end of this variable.

o **Translation**

The code parsed can be transformed into a user defined string. This string is passed back from sub-states to higher level calling states and can be interpreted by user sub-routines as required. For example, a subscripted variable reference can be regularized by translating each subscript to a hyphen. Thus $\wedge s1(s1a1, s1a2, x)$ would become $\wedge s1(-, -, -)$

If no translation is specified then the value passed back to the calling state is the same as the character string that has been parsed.

The keyword *null* has a special meaning. This translates the parsed string into a null string. This can be used to suppress the result of parsing a sub-state.

Translation can also be performed from within a user sub-routine by appending data to the variable *out*. NB manipulating the variable *out* does not affect the function of the translation field. If you want this to be suppressed then the transition should have a translation of *null*.

Editing Parser Diagrams

The screen in figure 5.7 is invoked when the Add/Modify option is selected. If a state name is entered in the Display field then the parser diagram for that state is displayed. This enables a change to be made in the context of a complete parser diagram, ensuring that transitions are defined correctly.

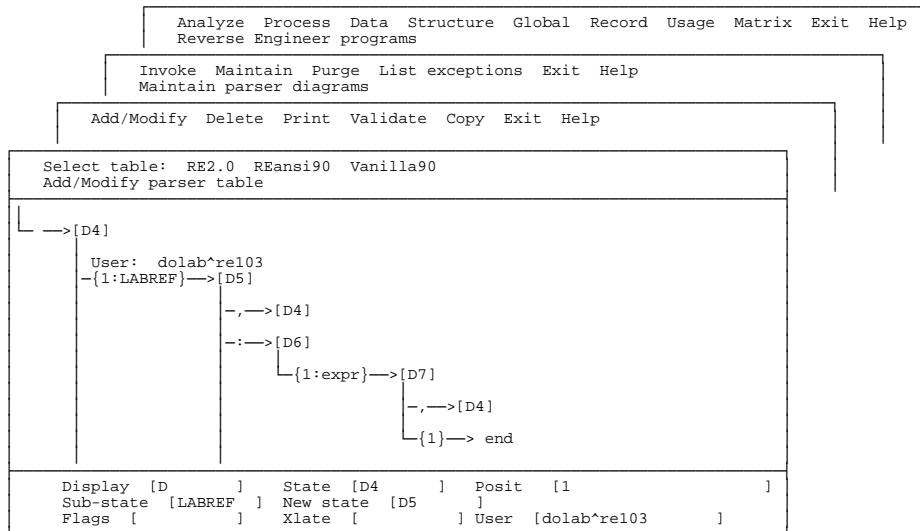


Figure 5.7 Parser Diagram Maintenance

By convention, states with lower case names are internal to the code analyzer. Because of this they cannot be modified. They can, however, be referenced by user defined states. If a state is created with the same name as an internal state the external state will take precedence. Thus an internal state can be overridden by a user defined state. The following internal states exist:

- expr parse an expression
- lit parse a string literal (excluding leading ")
- number parse a number (including decimal part and exponent)
- atom parse an expression atom
- anychar parse any single character excluding a space
- name parse an un-subscripted local variable name
- slash parse a backslash \'

The factory shipped parser tables may change from release to release of RE/m. For this reason, user modifications to the parser diagrams should be carefully controlled. It is strongly recommended that any new state created be pre-fixed with the character %. Future releases of RE/m will not use this name range and so will not conflict with user defined states. It is also recommended that changes be made in an empty parser table and that these are merged into a *copy* of one of the standard parser tables to create a working parser table. This will ease the process of upgrading parser table customization in future releases of RE/m.

o **Display**

This prompt determines which parser diagram is displayed in the window. If you want to display a parser diagram, you must enter the name of a state that exists in the parser table being edited. If you are creating a new parser diagram and do not wish to display anything then leave this field blank. The diagram that is displayed and the diagram that you edit do not have to be the same.

A diagram is defined as all of the states that are invoked from any given state. Therefore any state can be entered at this prompt and will result in a diagram showing what will be parsed from that state.

If the diagram is longer than the length of the screen then each page of the diagram will be displayed in turn. When the require page is displayed, select the Exit option at the Continue/Exit menu and you will be returned to the parser diagram edit window.

o **State**

Enter the name of the state that you wish to create or modify. The state does not have to exist, nor does it have to be visible on the display window in order to be permitted to modify it. As an aid to understanding what you are changing it is recommended that you use the display prompt to display the state that you are working on.

Once you have made several changes to a state it is advisable to refresh the display to review the results of your changes.

o **String / Posit**

This field toggles between prompting for Strings and Posits by pressing the return key.

If you wish to add or modify a string transition, toggle to the String prompt and enter the string to be parsed. If this transition already exists then the attributes of the transition will be displayed. If it does not exist then the fields in the edit window will remain empty.

If you wish to define a transition of a string comprising of a single space you will find that RE/m treats a single space as an instruction to delete the contents of a field. In order to cater for this special case, if you enter two spaces then this is interpreted as a single space.

The keyword *null* can be used to cater for the case where the end of the current input string (line of code or whatever) has been reached. When the end of the string being parsed is reached the parser will encounter an infinite number of nulls. Thus testing for a null string will cater for situations where the end of a line is expected.

If you wish to add or modify a Posit then toggle to the Posit prompt and enter the sequence number of the posit. If the posit already exists then the attributes of the posit will be displayed. If it does not exist then the fields in the edit window will remain empty.

Each transition in a parser diagram is uniquely identified by a combination of its source state and the string or posit from that state. It should be kept in mind, when parser table are merged, that each state and string/posit combination is in fact a separate and independent item.

o **Sub-state**

If a posit is selected then you will be prompted for a sub-state. This is the sub-tree that must be parsed in order to complete the transition from the current state to the next state.

If no sub-state is entered then the code analyzer will move to the next state without parsing any characters in the input file.

If the sub-state does not exist, this will be permitted, but a warning message will be displayed. In this case, you must remember to create the sub-state before you run the code analyzer on this table.

o **New State**

This field defines the state to which the parser will proceed if it successfully parses the string or posit being defined.

If the new state does not exist, this will be permitted, but a warning message will be displayed. In this case, you must remember to create the new state before you run the code analyzer on this table.

If the keyword *end* is entered then this will complete the parsing of a sub-state and return the parser to the calling diagram.

If you wish to delete a transition from a parser diagram then delete the contents of this field by entering a single space, followed by return. A *deleted* message will appear to indicate that the transition has been deleted.

o **Flags**

If you wish the transition to cause one or more flags to be set then enter the list of flags at this prompt.

Each flag is represented by a single character. Therefore in the following example three flags A, F and 4 would be set, not a single flag called AF4.

Flags [AF4]

Please note that the parser tables RE2.0 and REansi90 make use of the U flag to identify the end of a module of code.

o **Xlate**

This field is used to specify how the parsed string will be transformed.

- o If the field is left blank then the parsed string will be left unchanged.
- o If a value is entered then the parsed string will be replaced by the contents of this field.
- o If the keyword null is entered then the parsed string will be replaced by the null string.

- o **User**

This prompt allows a user defined sub-routine to be called after the successful parsing of any string or sub-state. The format is a valid entry-ref. For example:

```
dolab^re103
^test
msg^re312(x, "$Z")
```

Validation

The parser diagram validation facility performs some simple checks on the validity of the parser diagram set. After any major changes to the parser diagrams, this facility can be used to ensure the integrity of the diagrams. Any errors detected by the validation process should be corrected before any attempt is made to use the code analyzer.

The following validation is performed:

- o The existence of all sub-states is confirmed
- o The existence of all 'new states' is confirmed
- o Any state that is not used by another state, and is therefore redundant, is reported

The fact that the set of parser diagrams are valid does not indicate that they will correctly parse a given syntax, only that they are internally consistent. A modified parser diagram should be tested in a controlled way before it is used generally.

Process Catalog

Contents

General Procedure	47
System/Group Catalog	48
Transaction Catalog	49
Program Catalog	51
Module Catalog	53
Module Allocation	54

The Process catalog provides the facility to document the programs and modules created by the code analyzer and to create and document higher level processes such as transactions and systems. The process catalog consists of five levels of process (fig 6.1). The two lowest levels, programs and modules are created automatically by the code analyzer. The two highest levels, systems and groups (sub-systems) must be created and maintained manually. The transaction level can be created manually or automatically, depending on whether the programs to be analyzed are grouped using a rigid naming convention.

The repository maintains links between the five levels so that a process at any level can be defined as a member of a higher level and as a set of lower level processes. There is one exception to this, however. Transactions are made up of linked sets of code modules rather than sets of programs. This is to enable a logical structure to be applied to unstructured groupings of code into programs. Chapter 6, Transaction catalog, describes the relationship between transactions and modules in more detail.

Cataloging of an application requires a significant amount of manual effort (approximately two man days per one hundred routines). If your application uses a regular routine naming convention then the benefits of cataloging are very small compared to just using the repository wildcard selection mechanisms. It is therefore recommended that cataloging only be performed for applications that do not have a regular and consistent routine naming convention.

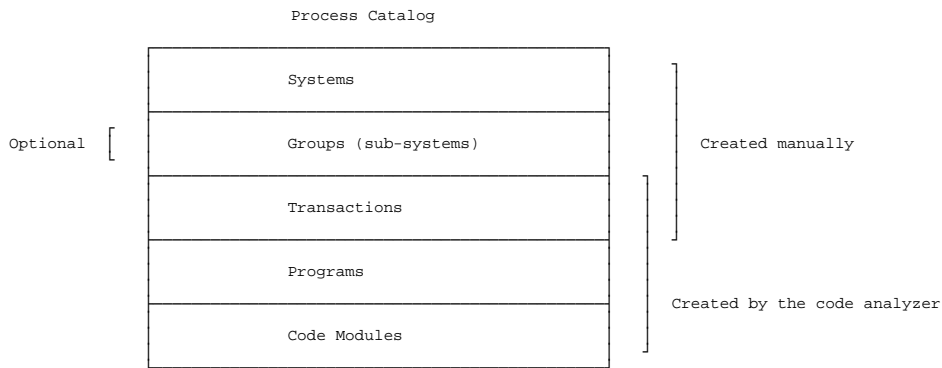


Figure 6.1 The process catalogs

Each entry in the process catalog is called a process. Documentation can be recorded in the catalog against any process. The following information can be entered:

- o **Process Reference**

An eight character unique reference that identifies the process.

- o **Description**

A narrative description of the process, up to forty characters in length.

- o **Document Reference**

A reference to an external piece of documentation, such as a user manual or functional specification, that further describes the process.

General Procedure

- o **Manual Cataloging**

The first step in defining a complete process catalog manually is to define the top two levels, systems and groups (sub-systems). The second level is optional, but is a useful way of breaking up a large system into more manageable sub-systems.

The second step is to reverse engineer the selected programs. This will automatically create the two lowest levels of the catalog. The analysis tools can all be used at the program and module level and so it is not necessary to create any other process objects within the catalog. However, creation of higher level objects can provide significant benefits by enabling the analysis of a complete system or sub-system, consisting of many programs, in one step.

The third step is to define the transactions that make up each system. A transaction is a set of code modules that are all called from each other to perform a single task at a single time. As each task is defined, the code modules that comprise that transaction are identified. An allocation analysis facility is provided to help you to allocate code modules to transactions. Once all code modules have been allocated (except for genuinely un-used modules) the creation of the process catalog is complete.

When one or more programs are modified and re-analyzed into the repository, the module allocation facility should be used to check that each code module is still correctly allocated.

o Automatic Cataloging

If the automatic cataloging option of the Code Analyzer is to be used then only the first two steps described above need to be carried out. Step three is performed automatically if the Auto-catalog option is selected in the code analyzer.

Once a set of programs has been analyzed using the Auto-catalog option it is then possible to alter and refine the resultant set of transactions in the same way that manually created transactions can be modified.

System/Group Catalog

The System/Group catalog provides a way of defining systems or applications to the repository. Each system represents a set of transactions that are related to each other either directly, by program calls or indirectly via the database.

Groups are an optional way of grouping transactions within systems. Groups can be used to make a large system more manageable by splitting it into sub-systems or they can be used to represent an arbitrary categorization of transactions. For example, Update and Enquiry or Batch and On-line transactions could be assigned to different groups.

Analyze Process Data Structure Global Record Usage Matrix Exit Help	
Process Catalog	
Add/Modify	Delete Show Print Exit Help
Maintain System/group Catalog	
System	[SALES]
Description	[Sales Analysis System]
Document Ref	[SAS.DOC/SPEC]
Group (1)	[MFM] [Master File Maintenance]
Group (2)	[DE] [Data Entry Facilities]
Group (3)	[BATCH] [Batch Processing Suite]
Group (4)	[MIS] [Management Information]
Group (5)	[] []

Figure 6.2 System/Group Catalog Maintenance

Transaction Catalog

The transaction catalog defines the individual units of processing within a system. Each transaction represents one or more pieces of code that, when executed, perform a definite function within a system. Examples of transactions are:

Customer Account Maintenance

Admit Patient

Order Entry

Print Customer Account Details

Reservation Enquiry

In a well structured system a transaction will correspond to one or more M programs. In a poorly structured system a transaction may correspond to arbitrary pieces of code scattered among many different programs, and shared by many processes. The transaction catalog provides a mechanism for allocating all the code within a system to its corresponding transaction.

The M code corresponding to a transaction is identified and defined in the repository by means of entry points. An entry point is the module in a program to which a call is made in order to perform a specific function. Often, the entry point is the first module of a program, but this is not always the case; a program may have several different entry points for different reasons, or sub-routines may have been placed at the top of the program to improve performance. Programs may also have additional entry points for indirect calls from utility sub-routines (eg Screen field validation). Likewise there may be entry points representing indirect returns from common overlays.

A frequently used technique is to copy common sub-routines into the programs where they are called from in order to avoid the performance costs of making a call to the sub-routine in a different program. A transaction can be defined as consisting of all code modules that have a particular name. This will automatically group all copies of the same sub-routine under one entry in the repository. There are two benefits in doing this. Firstly the multiple copies of the same piece of code can be considered as one object for impact analysis purposes, and secondly because the sub-routines are excluded from them the function and content of the main transactions will be clearer.

Having associated a transaction in the repository with its entry points, RE/m is able to automatically determine all the code modules that can be executed from those entry points. RE/m can do this even if many program overlays or sub-routine calls are involved. In this way specification of, in most cases, a single entry point can allocate all the code modules called by a transaction.

Figure 6.3 shows the screen that is used to maintain the transaction catalog within the repository. When building the transaction catalog, the following points should be noted:

- o All transactions must belong to a system that is defined in the repository.
- o Transactions may, optionally, be assigned to a sub-system within that system.
- o The transaction reference must be unique across the whole repository.
- o It is not necessary to define any entry points at the time the transaction is first entered. It is therefore possible to document a system's transactions before the corresponding entry points have been identified.
- o In order to get the best results from the analysis tools at and above transaction level, every code module in the repository (except for modules that are genuinely unused) should be allocated to *one* and *only one* transaction.
- o Use the Module Allocation facility in conjunction with the Transaction Catalog maintenance facilities to help identify modules that have not been allocated and modules that have been allocated to more than one transaction.

Analyze Process Data Structure Global Record Usage Matrix Exit Help	
Process Catalog	
System/Group Transaction Program Module Allocation Exit Help	
Maintain Transaction Catalog	
Add/Modify Delete Show Print Exit Help	
Transaction	[SMB]
Description	[Customer Account Maintenance]
System	[SALES] Sales Analysis System
Group	[]
Document Ref	[]
Entry point (1)	[a200^smb1]
Entry point (2)	[^smb2]
Entry point (3)	[*^smb3]
Entry point (4)	[]
Entry point (5)	[]
Entry point (6)	[]
Entry point (7)	[]
Entry point (8)	[] Updated

Figure 6.3 Transaction Catalog Maintenance

Any module in a program, that has been analyzed, can be specified as an entry point. Calls to 'Unknown Modules' can also be specified as transaction entry points. If the first module in a program is an entry point then it is sufficient to enter just the program name (preceded by an up-arrow).

If an asterisk is entered in place of a module name, then all unallocated modules within the specified program that are not called from other modules will be allocated to the transaction. If a module is called from another module it will not be allocated to the transaction automatically because it would normally belong to the same transaction as the module that calls it.

o **Common sub-routines**

Where common sub-routines are included in-line within more than one program all occurrences of the sub-routine can be cataloged under one transaction. This is achieved by identifying the name of the entry point(s) into the sub-routine and defining this to a transaction as an entry point. To indicate that the entry point is a common sub-routine that may occur in any program the name of the module should be suffixed with a wild-card routine name (eg DATE^*).

If a program contains a module with the same name as a common sub-routine but the module is not actually a copy of the common sub-routine, then it can be excluded by specifically declaring the module as an entry point to the transaction that it actually belongs to.

Program Catalog

The program catalog is a list of all the programs that have been analyzed by the code analyzer and loaded into the repository. The catalog also contains entries for programs that are not in the repository but are referenced by programs in the repository. The only way to add an entry to the program catalog is to run the code analyzer on the program's source code.

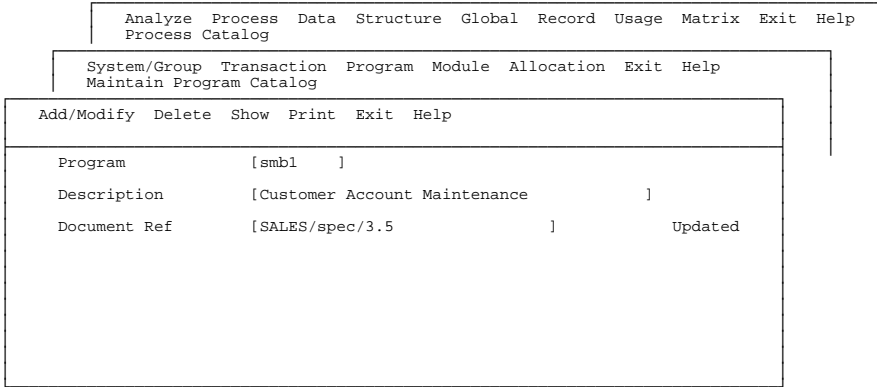


Figure 6.4 Program Catalog Maintenance

The Program Catalog screen (figure 6.4) is used for the following purposes:

- o To add or modify a description, or document reference, for a program in the repository.
- o To delete programs that are no longer used from the repository.
- o To display or print a catalog of all programs contained in the repository.
- o To display or print a catalog of all programs that are used by a system.

If a program is deleted from the catalog, all documentation in the repository, at program or module level, for that program will be deleted. This includes module descriptions and data documentation. If a program has been changed, it should *not* be deleted from the repository before being re-analyzed, otherwise any manually entered documentation for that program will be lost.

When a program catalog is printed, RE/m will prompt for a system reference. If no system reference is supplied, the program catalog will include every program in the repository. If a system reference is entered, then only those programs that are used by that system will be listed.

If a program is listed in a program catalog with a description of 'Unknown Program' then it is not in the repository but is referenced by one or more programs that are in the repository. Some of the analysis tools can be used on unknown programs. For example, Process Usage will show all the modules that call an unknown program.

Analyze Process Data Structure Global Record Usage Matrix Exit Help Process Catalog					
System/Group Transaction Program Module Allocation Exit Help Maintain Program Catalog					
Add/Modify Delete Show Print Exit Help Continue Exit					
Program	Description	Modules	Lines	Cmds	
init	Initialize screen	1	28	16	
lineup	Unknown Program				
menu	Sales System Menu	5	63	87	
scdef	Screen Definition	7	71	79	
scframe	Display Screen Frame	9	74	74	
screen	Screen Interpreter	29	385	372	
screen1	Screen Handler - Initialization	7	161	118	
scxlate	Set up terminal translation table	3	30	42	
sma1	Edit Sales Matrix	16	128	182	
smb1	Import Sales Data	19	129	254	
smb2	Merge Salesmen	8	85	124	
smb3	Region Matching	8	71	110	
smb4	Salesman Lookup	5	33	56	
smb5	Region Lookup	5	33	58	
smc1	Print Matrix	36	345	432	
smc2	Region lookup	7	44	87	
smc3	Area Lookup	7	69	97	
smd1	Daily Stats Help	5	30	56	
smel	Product File Help	6	48	62	

Figure 6.5 Program Catalog Report

Module Catalog

RE/m creates a module catalog for each program analyzed by the code analyzer. Entries in the module catalog can only be created or deleted by re-analyzing the program they belong to. Each module corresponds to a single piece of M code and is referenced by the first label belonging to that code.

The Module Catalog screen (figure 6.6) is used for the following purposes:

- o To add or modify a description and document reference for a module. A module description is automatically derived from any comments embedded within the program, however this can be manually overridden if required. The description is used on some reports and on the most expanded version of process structure diagrams.
- o To display or print a catalog of modules for a program.

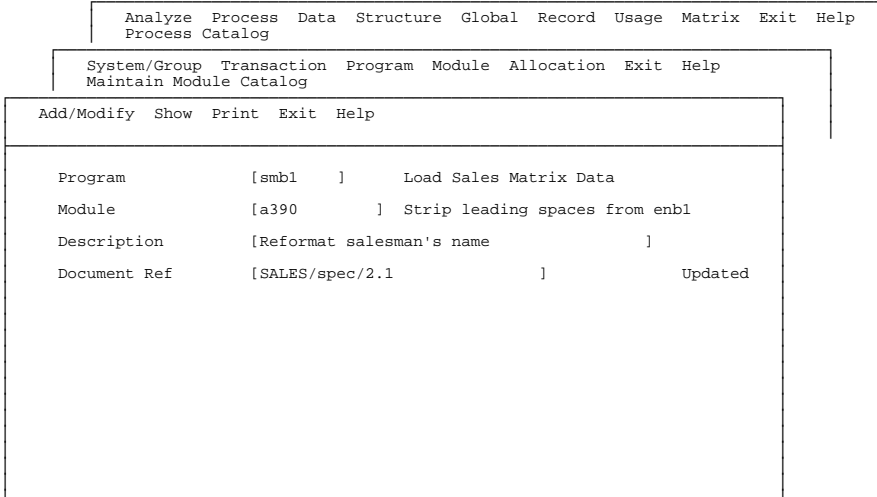


Figure 6.6 Module Catalog Maintenance

Where a program makes a call to a label in a program that has not been analyzed, a reference to that label is recorded in the repository. This is referred to as an 'Unknown Module'. If the function of the module is known a description can be provided, even if that program will never be analyzed. This is particularly useful for documenting calls to third party utility routines such as %D and %T.

Module Allocation

The Allocation option, on the Process Catalog menu, provides a set of analysis tools to help allocate modules to transactions. The objective, when defining the transaction catalog is to allocate all modules to *one and only one* transaction. The module allocation analysis displays assist in this by reporting:

- o Any module that is not allocated to a transaction.
- o Any module that is allocated to more than one transaction.
- o How modules are actually allocated, either individually or for a transaction or program.

The Module Allocation screen (figure 6.7) allows a module allocation analysis to be performed at four different levels:

- o **Repository**

All modules within the selected programs are analyzed. Modules are only reported if they are allocated to more than one transaction or are not allocated to any transaction.

- o **Transaction**

All modules that have been allocated exclusively to a transaction are displayed. If a module is allocated to more than one transaction then all the transactions that it belongs to will be listed.

- o **Program**

All modules belonging to a program are listed showing which transactions they have been allocated to. In a well structured system all modules in a program would belong to the same transaction. In a poorly structured system this may not be the case.

- o **Module**

All the transactions that own the specified module are listed.

Analyze Process Data Structure Global Record Usage Matrix Exit Help			
Process Catalog			
System/Group		Transaction	Program
Module Allocation Analysis		Module	Allocation
Repository		Transaction	Program
Display module allocation for a program		Module	Exit Help
Module	Transaction	Program	Module
a100^smb1	SMB	Customer	Account Maintenance
a200^smb1	SMB	Customer	Account Maintenance
a230^smb1	SMB	Customer	Account Maintenance
b100^smb1	SMC	Customer	Credit Check
b120^smb1	SMC	Customer	Credit Check
b900^smb1	-		
z100^smb1	SMB	Customer	Account Maintenance
	SMC	Customer	Credit Check
	SMD	Customer	Enquiry
z900^smb1	-		
z950^smb1	-		

Figure 6.7 Module Allocation Analysis

o **Unallocated Modules**

A module can be unallocated for a number of reasons. Before it can be correctly allocated to a transaction its usage should be investigated. The RE/m Process Usage analysis tool can be used, at the program or module level, to determine where and how a module is actually used.

If a module is genuinely unused then it need not be allocated to a transaction. However this does mean that it will not be included in any analysis above program level. If modules are temporarily unused, or may be used at some time in the future, then they can be allocated to a bucket transaction. This ensures they are included in any analysis of a system but ensures that they are easily distinguished from modules that really are used.

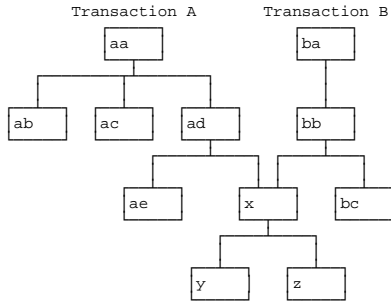
A module may appear to be unused if it is invoked and run directly from programmers mode, from a batch scheduler or from a tied terminal. User and operational procedures may need to be investigated to identify this type of usage.

A module may appear to be unused if it is called using indirection. Similarly, if a module is the indirect return point for a common overlay, there may be no explicit usage of the module. Detailed inspection of the program code may be necessary to identify how such a module should be allocated.

o **Multiple Allocation**

RE/m only allows a module to be directly allocated to one transaction. Despite this a module may become owned by multiple transactions. Figure 6.8 shows how this can happen. When a module is called by one other module, it is owned by the same transaction as the calling module.

If a module is called by several other modules, providing the calling modules are all owned by the same transaction, the called module will also be owned by that transaction. If, however, a module is called by several other modules and these calling modules belong to different transactions, then the called module will belong to all of these transactions.



In this example, modules from transaction A and transaction B both call module x. Because of this module x, and all modules called by x (ie y and z) are owned by both transactions.

Figure 6.8 Multiple Module Allocation

There are two ways of resolving a case of multiple allocation. If the called module is a common sub-routine or utility then it should be defined as a transaction in its own right (or part of a transaction representing a sub-routine library). Thus all transactions that call the module will be making a call to another transaction (figure 6.9).

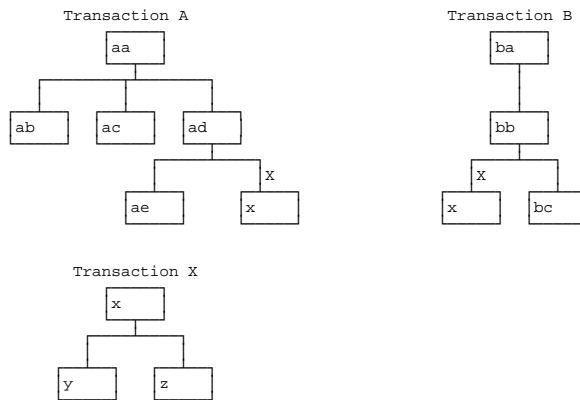


Figure 6.9 Module x as a common sub-routine

The second way of resolving multiple allocation is appropriate when the module should properly belong to one or other of the calling transactions. In this case the module can simply be made another entry point for the transaction it belongs to. All the other calling transactions will then treat this module in the same way as a call to a sub-routine or common module library (figure 6.10).

If multiple allocations are not resolved, then all the modules shared by both transactions may be reported more than once by the analysis tools.

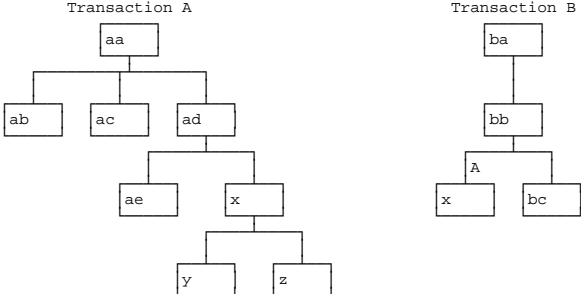


Figure 6.10 Module x as an entry point to transaction A

Data Catalog

Contents

Data Catalog Maintenance	60
Cataloging Guidelines	62
Data Catalog Reports	63

The data catalog enables documentation about local and global variables to be recorded in the repository. As with the process catalog, data can be documented at five different levels; System, Group (Sub-system), Transaction, Program and Code Module levels. Variables can also be documented at Repository level.

Entries in the data catalog are created automatically at program and module level, by the code analyzer. Once code modules have been allocated to transactions, the data catalog for a transaction will be defined implicitly. Descriptions for the data used by a process can be added to the catalogs using the data catalog maintenance screens.

Entries for local and global variables may be created manually for a process, even if the variable is not used by that process. This is so that, when a program is undergoing considerable change, new variables can be documented, within the repository, without the need to re-analyze the program immediately.

If process catalogs have not been set up for Systems, Groups and Transactions then variables can be documented at Repository, Program and Module levels only.

Data Catalog Maintenance

The data catalog is maintained from the Data sub-menu (figure 7.1). The process against which documentation is to be recorded is first selected and then the appropriate maintenance or reporting option is selected.

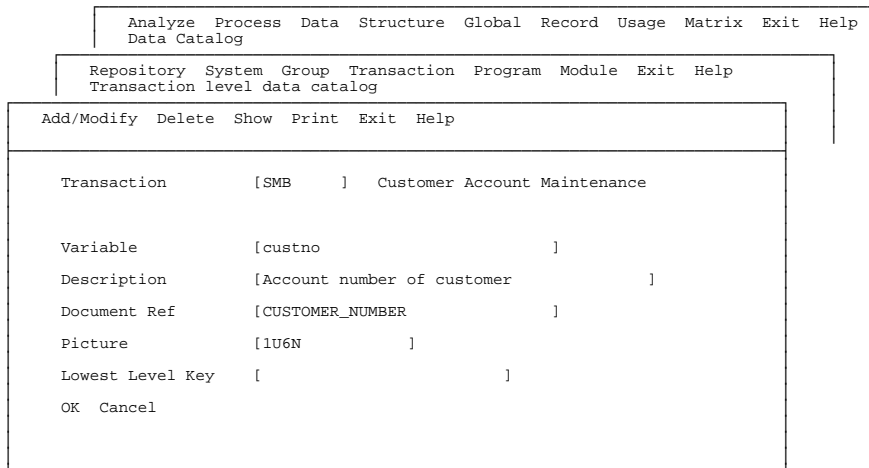


Figure 7.1 Data Catalog Maintenance

The following information may be recorded in the data catalog against a variable:

o **Description**

A narrative description of the purpose or data content of the variable.

o **Document Reference**

The reference of an external document that further defines the variable. This would most commonly be a reference to a data dictionary entry or a reference to a set of file layouts.

o **Picture**

The format of the data contained in the variable. This is a free text field allowing any convention to be used to describe the data format. Suggested conventions are either COBOL data division pictures or M pattern match expressions.

o **Lowest Level Key**

For subscripted local and global variables a description of the lowest level subscript can be entered to override the name of the key derived from the source code by the code analyzer. The value entered here will be displayed as the subscript name in global structure diagrams.

If a variable is documented against a high level process then its definition will apply to all subordinate processes that use that variable. Thus, variables that are common to a whole system need only be documented once despite being used in many different programs. If a standard variable is used in a non-standard way by a particular process, then the standard definition can be overridden for that process (whether it be a single module or a complete sub-system) simply by entering an alternative definition of the variable against the appropriate process.

Cataloging Guidelines

While the level at which variables should be documented is dependent upon the structure of the application being documented, the following guidelines may help to determine the best level:

- o Global Variables should be documented at repository or system level, unless they are, and will remain, specific to one sub-system or to one transaction.
- o The same global work file is often used by many transactions, but for different purposes in each case. If this is so, the work file should be documented against each transaction individually.
- o Local variables that are set up as part of the environment for the whole system (for example, standard delimiters, user ID's, etc) should be documented at repository or system level.
- o Local variables that are derived from globals as records and fields and only used in this way should be documented at the same level as the global from which they are derived.
- o Local variables which are used in the same way by one or more programs that call or overlay each other, and are all part of the same transaction, should be documented at transaction level.
- o Local variables that are used in a localized way within a single program or module should be documented at program or module level.
- o If the same variable is used for two different purposes in different modules of a single program, each usage should be documented against the appropriate module.

Data Catalog Reports

A print of the data catalog can be produced for any process in the repository. The format the printed data catalog is common to all process levels (figure 7.2), however the contents of the catalog depends upon the level of process being printed.

- o **Repository Data Catalog**

All and only those variables manually entered at repository level will be printed.

- o **System Data Catalog**

All and only those variables manually entered against the system will be printed.

- o **Group Data Catalog**

All and only those variables manually entered against the group will be printed. However, if the same variable has a definition at the system level, this definition will also be printed.

- o **Transaction Data Catalog**

All variables referenced by any module owned by the transaction will be printed, whether or not a description of the variable has been entered. Any variable that has been defined in the data catalog for the transaction, but is not used by any module in the transaction, will also be printed. These will be flagged as not used so that they may be deleted if the definition is no longer required.

If any variable is documented at a higher level than the transaction, its definition will also be printed on the report.

If any variable is documented at program or module level for any program or module owned by the transaction, then its definition will also be printed.

o Program Data Catalog

All variables reference within the program will be printed, whether or not a description of the variable has been entered. Any variable that has been defined in the data catalog for the program, but is not used by the program, will also be printed. These will be flagged as not used so that they may be deleted if the definition is no longer required.

If any variable is documented at transaction, group, system or repository level and that process owns any of the modules within the program, then its definition will also be printed.

If any variable is documented at module level within the program, then its definition will also be printed.

o Module Data Catalog

All variables reference within the selected module will be printed, whether or not a description of the variable has been entered. Any variable that has been defined in the data catalog for the module, but is not used by the module, will also be printed. These will be flagged as not used so that they may be deleted if the definition is no longer required.

If any variable is documented at program, transaction, group or system level and that process owns the module, then its definition will also be printed.

Analyze Process Data Structure Global Record Usage Matrix Exit Help Data Catalogs		
Repository System Group Transaction Program Module Exit Help Transaction level data catalog		
Add/Modify Delete Show Print Exit Help		
Variable	Description	Level
^en(-,-)	^en(area,region) Sales Region Record	S SALES
^en(-,-,-)	^en(area,region,territory) Sales Territory Record	R *
^tmp(-)	Sales records to be processed	T SMB
%	Standard Delimiter	S SALES
custno	Account number of customer	T SMB
option	Menu selection	P ^smb1
qty	Sales discount option	M disc^smb2
z	Sales quantity	T SMB
	Temporary counter	P ^smb1
^tmp(-,-)	Not used	
yrno	Not used	

Figure 7.2 Transaction level Data Catalog

Process Selection

Contents

Repository Level Selection	66
System Level Selection	68
Group Level Selection	69
Transaction Level Selection	70
Program Level Selection	72
Module Level Selection	73

All RE/m inquiries require a process selection to be made in order to select the subject of the inquiry.

The process selection dialog will vary depending upon the process level of the inquiry. This chapter describes the selection dialog for each process level.

Repository Selection

Repository level selection enables a set of one or more programs to be selected from the program catalog.

Figure 8.1 shows a typical repository level selection screen.

Analyze	Process	Data	Structure	Global	Record	Usage	Matrix	Exit	Help
Process Structure Diagram									
Repository	System	Group	Transaction	Program	Exit	Help			
Repository Structure Diagram									
Location	[DEV]								
Program(s)	[sla*]		145 programs selected			
No of significant characters to group by?	[16/8]								

Figure 8.1 Repository level selection screen

o Location

If VC/m is installed then the selection of items from the repository may be qualified by specifying a location. If a location is not entered then items will be selected from the repository as normal.

If a location code is entered then items will only be selected from the repository if they are present at the specified location. This enables inquiries on the RE/m repository to select the versions of programs that exist at any VC/m controlled location.

For example, if a problem is being investigated that has occurred at site x then selection of location x on this inquiry will make RE/m select only those versions of each program that are actually being used at location x.

If RE/m is being used as a quality control tool then selection of the TEST area for a new release of an application would enable RE/m to produce inquiries on just those versions of each program that are present in the test area.

o **Program(s)**

The standard Repository selection prompt enables one or more selections and de-selections to be made from the catalog of Programs within the repository.

The following inputs can be supplied:

- o A single program
- o A wildcard, to select all programs with the same name prefix. For example: SMA*
- o A range, to select all programs in a range. For example: SMA10-SMA20
- o De-selection of a single program. For example: -SMA10A
- o De-selection of a range. For example: -SMA*
- o De-selection of a range. For example: -SMA10-SMA20

Multiple combinations of the above can be made to refine the selection set if required.

The HELP key (or PF1, or F1 on a PC console, or ? if all else fails) will invoke a selection screen which will show in the left hand window the contents of the repository and in the right hand window the contents of the current selection set. See chapter 2, Selection windows, for details of how to use this selection window.

o **Number of significant characters to group by?**

This prompt allows the selected set of routines to be grouped into sub-sets based on matching name prefixes.

If, for example, the selection set contained programs called SMA1*, SMA2* and SMB1* then grouping this set by the first four characters of their names would result in three sub-sets called SMA1, SMA2 and SMB1. In contrast, if the grouping was by the first three characters then this would result in two sub-sets called SMA and SMB.

If grouping of programs is specified then RE/m will consolidate the results of the subsequent inquiry into the required sub-sets. This will result in information that is at a higher level of abstraction, making a trade off between detail and volume of information.

There are two parts to this prompt, separated by a slash. The first part refers to the program name prefix. The second part refers to the version number suffix and provides a way of consolidating or differentiating between different versions of the same program. In general, if the repository contains more than one version of each program, then each version can be shown as a separate item in the subsequent inquiry. This is the default and is achieved if all eight characters of the version suffix are treated as significant. If zero is entered then no part of the version suffix is treated as significant and so all versions of the same program will be consolidated together to produce one item in the subsequent inquiry.

System Selection

System level selection enables an inquiry to be performed on all transactions within a system.

Figure 8.2 shows a typical system level selection screen.

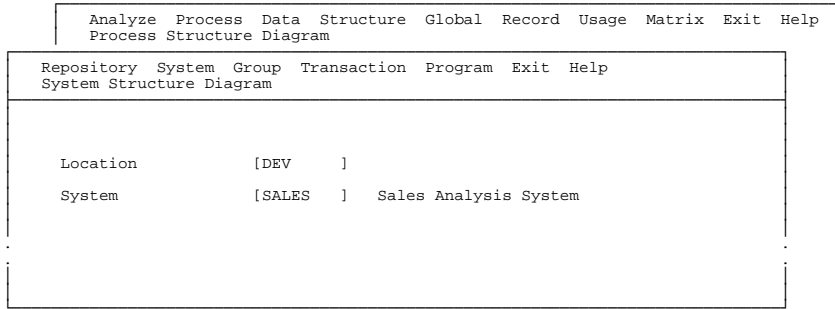


Figure 8.2 System level selection screen

o Location

If VC/m is installed then the selection of modules belonging to each transaction within the system may be qualified by specifying a location. If a location is not entered then items will be selected from the repository as normal.

If a location code is entered then items will only be selected from the repository if they are present at the specified location. This enables inquiries on the RE/m repository to select the versions of programs that exist at any VC/m controlled location.

For example, if a problem is being investigated that has occurred at site x then selection of location x on this inquiry will make RE/m select only those versions of each program that are actually being used at location x.

If RE/m is being used as a quality control tool then selection of the TEST area for a new release of an application would enable RE/m to produce inquiries on just those versions of each program that are present in the test area.

o System

The name of the system for which transactions are to be displayed should be entered in this field. The HELP key (or PF1, or F1 on PC Consoles, or ? if all else fails) will invoke a look-up window from which selection can be made using the cursor keys.

Group Selection

Group level selection enables an inquiry to be performed on all transactions that belong to a specific group within a system.

Figure 8.3 shows a typical group level selection screen.

Analyze Process Data Structure Global Record Usage Matrix Exit Help			
Process Structure Diagram			
Repository System Group Transaction Program Exit Help			
Group Structure Diagram			
Location	[DEV]		
System	[SALES]	Sales Analysis System	
Group	[MGMT]	Management Reports	

Figure 8.3 Group level selection screen

o Location

If VC/m is installed then the selection of modules belonging to each transaction within the group may be qualified by specifying a location. If a location is not entered then items will be selected from the repository as normal.

If a location code is entered then items will only be selected from the repository if they are present at the specified location. This enables inquiries on the RE/m repository to select the versions of programs that exist at any VC/m controlled location.

For example, if a problem is being investigated that has occurred at site x then selection of location x on this inquiry will make RE/m select only those versions of each program that are actually being used at location x.

If RE/m is being used as a quality control tool then selection of the TEST area for a new release of an application would enable RE/m to produce inquiries on just those versions of each program that are present in the test area.

o System

The name of the system for which transactions are to be displayed should be entered in this field. The HELP key (or PF1, or F1 on PC Consoles, or ? if all else fails) will invoke a look-up window from which selection can be made using the cursor keys.

o Group

The name of the group, within the above specified system, should be entered in this field. It is possible to select only those transactions that are not allocated to any specific group within the system by entering a hyphen in this field.

The HELP key (or PF1, or F1 on PC Consoles, or ? if all else fails) will invoke a look-up window from which selection can be made using the cursor keys.

Transaction Selection

Transaction level selection enables an inquiry to be performed on the programs or modules that belong to a specific transaction.

Figure 8.4 shows a typical transaction level selection screen.

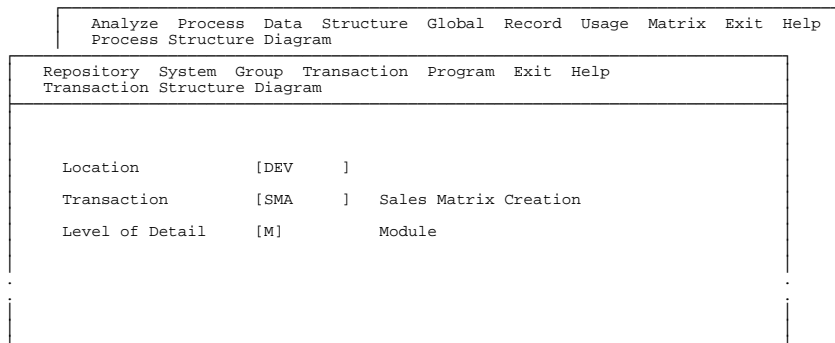


Figure 8.4 Transaction level selection screen

o **Location**

If VC/m is installed then the selection of modules belonging to the transaction may be qualified by specifying a location. If a location is not entered then items will be selected from the repository as normal.

If a location code is entered then items will only be selected from the repository if they are present at the specified location. This enables inquiries on the RE/m repository to select the versions of programs that exist at any VC/m controlled location.

For example, if a problem is being investigated that has occurred at site x then selection of location x on this inquiry will make RE/m select only those versions of each program that are actually being used at location x.

If RE/m is being used as a quality control tool then selection of the TEST area for a new release of an application would enable RE/m to produce inquiries on just those versions of each program that are present in the test area.

o **Transaction**

The name of the transaction should be entered in this field. The HELP key (or PF1, or F1 on PC Consoles, or ? if all else fails) will invoke a look-up window from which selection can be made using the cursor keys.

o **Level of Detail**

Transaction level inquiries can be displayed at one of two levels of detail. Program level will display all modules belonging to the transaction consolidated by the program to which they belong. Module level will show each module belonging to the transaction as a separate item in the inquiry.

Program Selection

Program level selection enables an inquiry to be performed on a single program.

Figure 8.5 shows a typical program level selection screen.

Analyze	Process	Data	Structure	Global	Record	Usage	Matrix	Exit	Help
Process Structure Diagram									
Repository	System	Group	Transaction	Program	Exit	Help			
Program Structure Diagram									
Location	[DEV]						
Program	[SMA10/2.6] Sales Matrix Input Routine						

Figure 8.5 Program level selection screen.

o Location

If VC/m is installed then only those versions of a program that exist at the specified location may be selected. If a single version location is entered then the program version will default to the version that is present at the designated location.

o Program

The name of the program should be entered in this field. If no explicit version number is entered there are multiple versions of the program in the repository (subject to any qualification by the location field) then the version with the highest version number will be selected.

The HELP key (or PF1, or F1 on PC Consoles, or ? if all else fails) will invoke a look-up window from which selection can be made using the cursor keys.

Module Selection

Module level selection enables an inquiry to be performed on a single module within a program. Figure 8.6 shows a typical module level selection screen.

Analyze	Process	Data	Structure	Global	Record	Usage	Matrix	Exit	Help	
Process Structure Diagram										
Repository	System	Group	Transaction	Program	Module	Exit	Help			
Module usage										
Location	[DEV]							
Program	[SMA10/2.6]							Sales Matrix Input Routine
Module	[INIT]							Initialization

Figure 8.6 Module level selection screen.

o Location

If VC/m is installed then only those versions of a program that exist at the specified location may be selected. If a single version location is entered then the program version will default to the version that is present at the designated location.

o Program

The name of the program should be entered in this field. If no explicit version number is entered there are multiple versions of the program in the repository (subject to any qualification by the location field) then the version with the highest version number will be selected.

The HELP key (or PF1, or F1 on PC Consoles, or ? if all else fails) will invoke a look-up window from which selection can be made using the cursor keys.

o Module

The name of the module should be entered in this field. Each module represents a paragraph of code within the repository. Each module is identified by the name of the first label within the module. The label corresponding to the start of the module is therefore the name of the module.

The HELP key (or PF1, or F1 on PC Consoles, or ? if all else fails) will invoke a look-up window from which selection can be made using the cursor keys.

Process Structure Diagrams

Contents

Diagram Notation	77
Screen Operation	79

The Process Structure Diagram facility is an analysis tool that gives a graphic interpretation of the internal structure of a process. For any process it shows the individual components of that process and the relationship between the components.

Process Structure Diagrams can be produced at five levels, and can either be viewed interactively on a VDU screen or printed. The five levels are as follows:

o **Repository**

The diagram shows the relationship of programs within a selected set of programs. The degree of detail can be altered by grouping programs beginning with the same characters together. For example, if the number of significant characters is set to four then all programs with names that have the same first four characters will be grouped into one box on the diagram.

o **System**

The diagram shows all transactions within a selected system and all control flows to transactions in other systems.

o **Group**

The diagram shows all transactions within a selected group (or sub-system). All control flows to transactions in other systems and groups are also shown.

o **Transaction**

The transaction structure diagram can be produced at module or program level. At module level all code modules invoked by the transaction are shown. Calls and transfers of control to code modules in other transactions are also shown.

At program level all programs containing modules belonging to the transaction are shown. Control flows to modules in other transactions are shown as control flows to the program in which the module occurs.

o **Program**

All code modules within the program are shown. Invocation of modules within other programs are also shown.

The analysis tools do not support the examination of the internal structure of a single module. Typically a code module should be sufficiently small that its structure is best understood by examination of the code directly.

Diagram Notation

The diagram notation used is an adaptation of the Segmented Level notation used for modular system design. Figure 9.4 gives an example of this type of structure diagram. The full diagram notation is as follows:

o Process

A rectangular box represents a process. The process may be a transaction, a program, a group of programs or a code module depending upon the level of the diagram.

Depending upon the display option, the box may contain just the process reference, or it may contain both the reference and the description of the process.

o Control Flows

A line between two processes represents a control flow. Control flows are caused by commands such as Do, Goto, Job and Zcall. Figure 9.1 shows the notation used to represent each type of control flow.

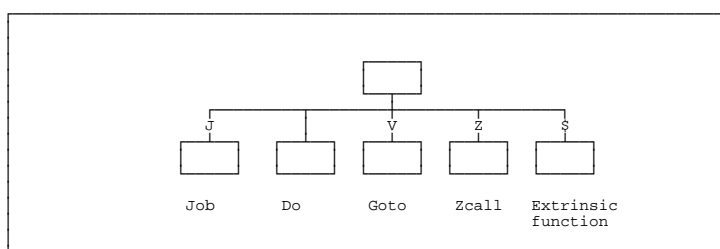


Figure 9.1 Control Flow Notation

o External Calls

When a call or transfer of control is made to a component of a process that is not part of the selection set being displayed, the control flow is flagged with the name of the external process that is referenced. The box will also be shown with double vertical edges to indicate that it is not decomposed in the diagram.

o Duplicate Occurrences

In order to achieve a compact representation, any process that appears on a diagram more than once will only be expanded in one place. Whenever it occurs again on the diagram it will be shown with a small rectangle in the top left hand corner. This indicates that it is decomposed elsewhere on the diagram.

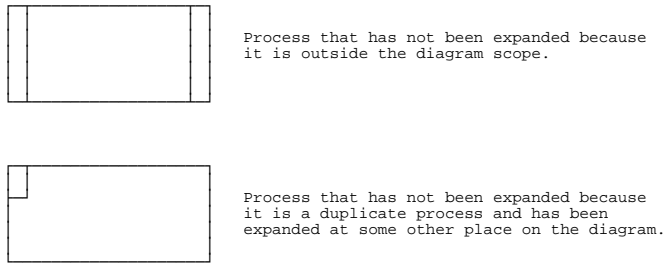


Figure 9.2 External call and Duplicate Occurrence Notation

o External Entry Point

If a process is called from one or more other processes that are outside the selection set being displayed then the process can be considered to be an external entry point into the selection set. This is represented by a tag in the top right-hand corner of the process box.

o Internal Entry Point

If a process is called from one or more other processes within the selection set (other than its immediate parent on the diagram) then a tag will be displayed in the top left hand corner of the process box. This indicates that the process is referenced more than once within the diagram. The other references to the same process will be represented as duplicate processes and will not have been expanded at the point where they are displayed.

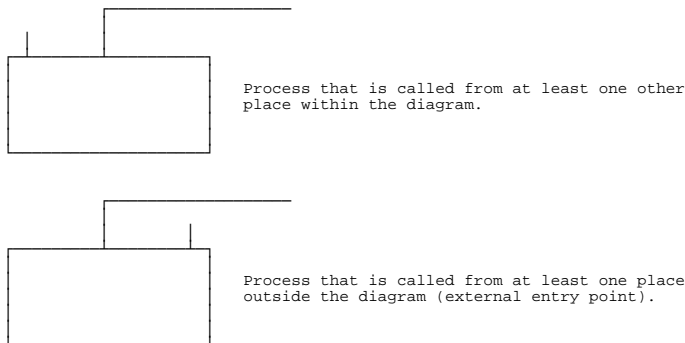


Figure 9.3 Internal and External Entry Points

Screen Operation

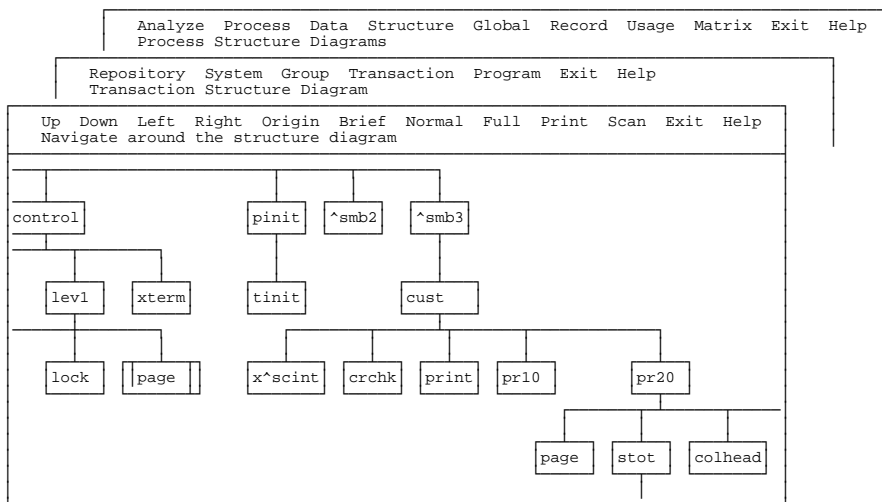


Figure 9.4 Process Structure Diagram

o **Up Down Left Right**

When a diagram is larger than the screen, the commands Up, Down, Left and Right enable different parts of the diagram to be examined, by moving the diagram in the selected direction. Each movement is approximately one eighth of the size of the screen. If the appropriate key is held down the diagram will move more quickly.

On most terminals, the arrow keys may be used to perform the same function as Up, Down, Left and Right.

o **Origin**

The Origin command returns the display to the first object on the top line of the diagram.

o **Brief**

The brief display option only shows the module label or process reference of each component of the diagram. It is very compact and enables more of a diagram to be seen at one time, than the other display options.

o **Normal**

This option selects the normal display format. This format shows the module label or process reference of each diagram component, and each component is enclosed within a box, giving a clearer representation at the expense of conciseness.

o **Full**

The Full display format shows both the module label or process reference and the description of the component within a rectangular outline. This format gives the most detail but, consequently, is not concise.

o **Print**

The Print command prompts for an output device, which may be a printer or an ASCII format sequential file (subject to Operating System support), and prints the complete structure diagram in the format of the currently selected display option.

o **Scan**

The scan option provides a way of searching the diagram for the occurrence of a string. This can be used to find the location of an object by entering its name.

The scan will search for any partial or full match on both the name of the object and its description. The description will be searched even if the brief display option is currently selected.

When an object has been found that matches the search string it is highlighted and if it is not currently displayed the display window is moved so that it can be seen.

Global Structure Diagrams

Contents

Diagram Notation 82
Screen Operation 84

The Data Structure Diagramming analysis tool produces a graphic representation of the structure of local or global variable arrays as used by a particular process. This tool can be used to identify which nodes of a global file are created and used by a system. It can also be used to identify the sub-set of a file structure that is used by a particular sub-system, transaction, program, module or selected set of programs.

Knowledge of the structure of the data represented within any database is vitally important to the full understanding of the design of a system. Without an understanding of the data structures within a system it is difficult to determine whether a new requirement can be met with the existing data structures or not. If data structure changes are required then the impact of the change will be more significant than if no structural changes are necessary. The data structure diagrams produced by RE/m meet this need by representing global nodes as hierarchical structures. This representation can be mapped onto a network style Entity/Relation model, by further manual analysis, if required.

Diagram Notation

The diagramming notation used for data structure diagrams is a hierarchical representation of M local and global variable structures, with each node being represented by a rectangle and each subscript level being represented by a branch of the hierarchy.

o **Record Node**

A large rectangle represents a local or global variable node that contains data. In conventional DBMS's this is equivalent to a record. If a description has been entered into the data catalog for the node then, depending upon the display option, this description will be printed within the rectangle .

o **Pointer Node**

A small rectangle represents a pointer only node in a data structure. These nodes may or may not be referenced within an application. If they are referenced, they will never be read or set but their existence may be determined using functions such as \$data and \$order.

Pointer nodes should not be confused with nodes that are null, such as would be used for alternate indexes, where the node does exist, is read and set by the application, but contains null data. Null nodes are represented as Record Nodes in this notation.

o Subscripts

Each subscript level, of a variable reference, is represented by a branch on the hierarchy diagram. Fixed subscripts are treated as distinct from subscripts accessed using a variable or compound expression. Fixed subscript branches are labeled, on the diagram, with the value of that subscript.

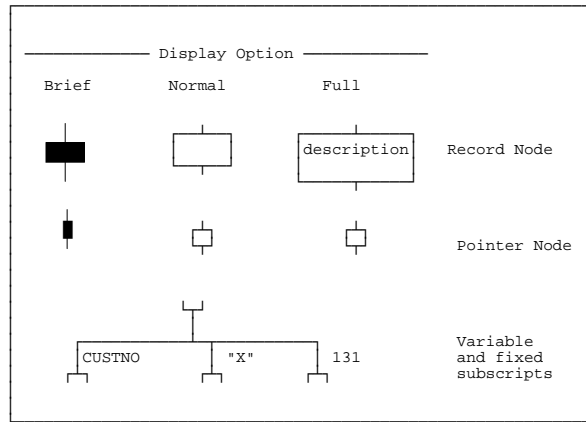


Figure 10.1 Data Structure Diagram Notation

Screen Operation

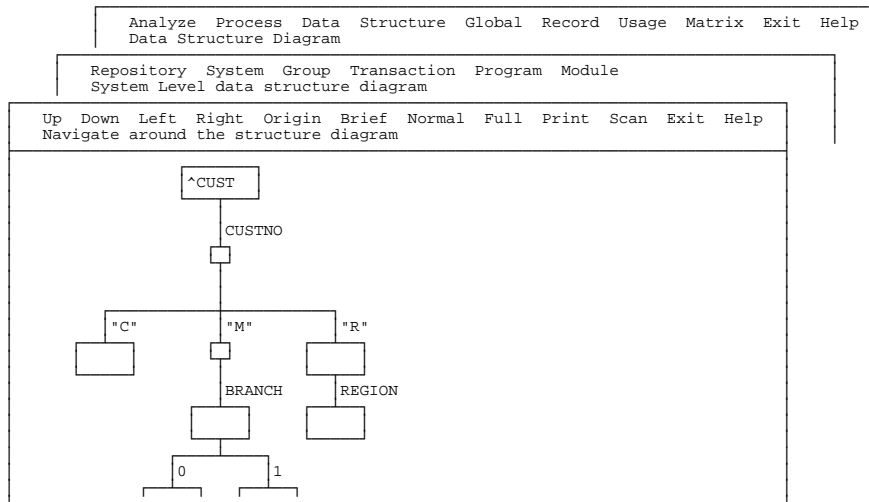


Figure 10.2 Data Structure Diagram Screen

o Up Down Left Right

When a diagram is larger than the screen, the commands Up, Down, Left and Right enable different parts of the diagram to be examined, by moving the diagram in the selected direction. Each movement is approximately one eighth of the size of the screen. If the appropriate key is held down the diagram will move more quickly.

On some terminals, the arrow keys may be used to perform the same function as Up, Down, Left and Right.

o Origin

The Origin command returns the display to the top-left hand corner of the diagram.

o Brief

The brief format only shows the structure of the local or global variable and any fixed value subscripts. It is very compact and enables more of a diagram to be seen at one time than the other display options.

- o **Normal**

This format shows the structure of the local or global variable with each pointer node and record node represented as a rectangular box. This gives a clearer representation at the expense of conciseness.

- o **Full**

The expanded format shows the structure of the local or global variable and, if a description has been defined within the data catalog, each record node will include its associated description within the box icon. This format gives the most detail but, consequently, is not concise.

Note that for pointer nodes, if a description has been entered in the data catalog, it will not be displayed on the diagram.

- o **Print**

The Print command prompts for an output device, which may be a printer or an ASCII format sequential file (subject to Operating System support), and prints the complete structure diagram in the format of the currently selected display option.

- o **Scan**

The scan option provides a way of searching the diagram for the occurrence of a string. This can be used to find the location of an object by entering its name.

The scan will search for any partial or full match on both the name of the object and its description. The description will be searched even if the brief display option is currently selected.

When an object has been found that matches the search string it is highlighted and if it is not currently displayed the display window is moved so that it can be seen.

Process Usage Matrices

Contents

Notation 89

Screen Operation 90

The Process Usage Matrix shows how a process within a system is used and how it uses other processes. It also shows how the components of a process relate to each other.

This matrix is used to identify the impact of making a change to a particular process or component of a process. The matrix can be produced at all five repository levels or for any selected set of programs enabling both high and low level impact analysis to be performed. For example, at system level the interfaces between two systems can be identified while at module level the places from which a particular code module are called can be established.

- o **Repository Level**

When a matrix is created at repository level one or more programs can be selected either by entering the individual names of the programs or by using a wild-card selection.

The matrix rows and columns can be grouped by any number of significant characters to provide high or low level detail. For example, if three significant characters are selected then the matrix will group all programs that begin with the same three characters under one row or column.

- o **System Level**

If a System level matrix is requested then the matrix will show the calling relationships between all the transactions that belong to that system.

- o **Group Level**

When a group (sub-system) is selected then the matrix will show the calling relationship between the transactions that belong to that particular group.

- o **Transaction Level**

The transaction level matrix can be displayed at two levels of detail, program or module. The module level of detail shows the calling relationship between all modules that are owned by the selected transaction.

The program level of detail shows the calling relationships between all the modules that are owned by the selected transaction but the information is consolidated by program in order to produce a more compact display.

- o **Program Level**

The program level matrix shows the calling relationships at module level between all the modules that make up the program.

o **Module Level**

The module level matrix shows all the calling relationships between a single module within a program and all other modules within the repository.

Notation

The matrix is made up of rows and columns, each representing a component of the selected process or a component that uses or is used by the selected process. Each cell in the matrix describes the way that the corresponding components call each other.

The rows and columns of the matrix are both split into two parts, separated by a blank row and column. This divides the overall matrix into four areas (figure 11.1). The top left area contains all the component processes that are part of the selected process and therefore describes the relationships between the components within the selection set. The top right area contains all the external processes that are called by members of the selection set. The bottom left area contains all the external processes that call members of the selection set. The bottom right area is always blank.

Calling Transaction		Transaction					Called		
		A	B	C	D	E	X	Y	Z
A			D		G				G
B							\$		
C							\$		
D					D				
E		G							
P									
Q		D							
R				D					

Figure 11.1 Process Usage Matrix

The cell contents describe how the calling module transfers control to the called module. There are five different ways that a process can be invoked, as follows:

- D DO The process is invoked with a DO command, on completion control returns to the calling process.
- G GO Control is transferred with a GO command. There is no automatic return.
- \$ Extrinsic A process is invoked using an extrinsic function call. On completion function control returns to the calling process.
- J JOB The subject is invoked as a separate concurrent process. The invoking and the invoked processes will both continue to execute.
- Z Zcall A call is made to a function written in another language by means of the Zcall function. The name of the function is represented as a module.

Screen Operation

Analyze Process Data Structure Global Record Usage Matrix Exit Help Process Usage Matrix												
Repository System Group Transaction Program Module Exit Help System Usage Matrix												
Up Down Left Right Origin Print Screen Cursor Tab Exit Help Navigate around the matrix												
System SALES	Transaction called											
Calling Transaction	SMA	SMB	SMC	SMD	SME	SMF	SNA	SNB		LIB0	LIB1	SCR
SMA		G			D					\$	\$	D
SMB			G	D	D					D\$	\$	D
SMC	G	G								\$	\$	D
SMD					D		J	J				D
SME												D
SMF											\$	D
SNA										D\$		D
SNB										D\$		
MENU	D	D	D			D						
BATCH							DJ	DJ				

Figure 11.2 System Level Usage Matrix

- o **Up Down Left Right**

When a matrix is larger than the screen, the commands Up, Down, Left and Right enable different parts of the matrix to be examined, by moving the matrix in the selected direction. The effect of the movement commands depends upon the selected display mode. Consult the screen, cursor and tab commands for more information.

On most terminals, the arrow keys may be used to perform the same function as Up, Down, Left and Right.

- o **Origin**

The Origin command returns the display to the top-left hand corner of the matrix.

- o **Print**

The Print command prompts for an output device, which may be a printer or an ASCII format sequential file (subject to Operating System support), and prints the complete Usage matrix.

- o **Screen (Default)**

The screen option selects the screen paging mode. In this mode when the arrow keys are pressed the matrix display will pan in the direction indicated.

- o **Cursor**

If the cursor mode is selected highlighted vertical and horizontal cursor bars are displayed on the matrix. Using the arrow keys will move the cursor bars in the direction indicated by one row or column at a time.

- o **Tab**

If the tab mode is selected then cursor bars will be displayed, as for the cursor mode. However, when the arrow keys are used to move the cursor bars instead of moving one column or row at a time the cursor bar will tab to the next cell that contains an entry. This enables the entries in sparse matrices to be viewed quickly and accurately.

Process/Data Matrices



Contents

Notation 97

Screen Operation 99

The Process/Data Matrix shows the interaction between a process and the data that it uses. This enables the impact of changes to data usage, format or structure to be identified. The matrix can also be used to perform various types of completeness and consistency checking by enabling the usage of data to be inspected across a whole system.

The matrix can be produced at five different levels; repository, system, group, transaction and program. The contents of the matrix varies depending upon the level of the selected process. The table in figure 12.1 summarizes this.

Process Level	Column Contents	Row Contents
Variable	Program groups	Global, local or both
Repository	Program groups	Global, local or both
System	Transactions	Global, local or both
Group (sub-system)	Transactions	Global, local or both
Transaction	Programs or Modules	Global, local or both
Programs	Modules	Global, local or both

Figure 12.1 Matrix Contents Summary

For each repository level there are a number of selection and display options that can be used to refine the interrogation of the repository. Figure 8.2 shows a sample dialog for the repository level selection.

```

Analyze Process Data Structure Global Record Usage Matrix Exit Help
Process/Data Matrix
-----
Variables Repository System Group Transaction Program Exit Help
Repository level Process/Data matrix
-----
Location          [          ]
Program(s)        [          ]      582 programs selected
No of significant characters to group by? [16/8]
Local variables, Global variables or Both? [B]
Variable(s) [          ]      All variables selected
Number of subscript levels? [  ] All levels
Operations to include [RSTDKXLUIQNY]
    
```

Figure 12.2 Repository level selection dialog

o **Process Selection**

The prompts for the selection of the process axis (columns) of the matrix depend upon the selected process level. In most cases there is a prompt for a single object of the appropriate type (eg System or Program). However, if the Repository level is selected then there will be an additional prompt for the number of significant characters of the program name to group programs together into one column.

If the Variable level selection is used then the programs displayed are determined by the variables selected. Only those programs that reference one or more of the selected variables will be included in the matrix.

o **No of significant character to group by**

This is only prompted for repository level selections. The programs that appear on the matrix will be grouped into columns by the matching characters of the program name. If the default is selected then each column will contain an individual program.

If multiple versions of each program are represented in the repository then each version of the same program can be represented individually or all versions of a program can be consolidated together by entering eight or zero respectively in the second part of this field.

o **Level of Detail**

This is only prompted for Transaction level selections. The columns of the Transaction level matrix can show either individual code modules within a transaction or can consolidate the information by program to produce a summary level display. If the program level of detail is selected then each column will contain a single program and the content of the column will comprise of the operations performed by only those modules in the program that are owned by the selected transaction.

o **Local variables, Global variables or Both?**

The matrix display can be set to include only Global variable references, only Local variable references or Both local and Global references. The selected set can be further constrained by entering a specific list of variables at the next prompt.

o **Variable(s)**

The data axis of the matrix can be constrained by entering a specific list of variables that are required. This prompt repeats until a null entry is input, allowing multiple selections to be made.

There are a number of selection options that can be entered:

- o A single local or global name can be entered. If the variable is subscripted then by default all subscripted nodes for that variable will also be selected.
- o A wild-card selection can be made. For example, entering SCR* will select all variables beginning with SCR.
- o A range of variables may be selected. For example, entering A-C will select all variables that have names between A and C.
- o De-selection by entering any of the above preceded with a minus sign. For example, -SCR* will de-select all selected variables starting with SCR.

If no selection is made then the matrix will include all variables referenced by the selected process.

o **Number of subscript levels?**

When high level matrices comprising of global variables are produced it is sometimes useful to consolidate the data axis by grouping all references at lower subscript levels together. The number of subscript levels that variables are consolidated by can be selected at this prompt. If 0 is entered then all subscripted nodes for each variable will be consolidated into one row under the name of that variable.

If no value is entered at this prompt then the default will be to display all levels by not consolidating the rows at all.

o Operations to include

This prompt allows the user to select a sub-set of the operations that will appear on the matrix. In some cases it is useful to be able to produce a matrix that contains a sub-set of the operations. For example, a matrix can be produced that shows only update operations (S and K), or a matrix might be required to help investigate a locking problem, in which case only Locks and Unlocks would be required.

The default is that all operations will be included in the matrix.

Notation

The matrix consists of columns representing the components of the selected process and rows representing the individual data items used by the process. Each cell in the matrix contains codes that indicate the manner in which the process uses the item of data.

The following legend is used in the matrix cells to represent how the data in each row is used by the process in each column:

Code	Meaning	Code	Meaning
R	Referenced (used)	L	Lock
S	Set	U	Unlock
I	Input (READ command)	T	Traverse (\$order, \$next)
O	Output (WRITE command)	D	Status (\$data)
K	Kill	N	New
X	Exclusive kill	Y	Exclusive new

Certain commands are un-specific about the variables they apply to (argumentless new, lock and kill). These are shown in the matrix against a pseudo variable called '*'. This should be interpreted to mean potentially all variables.

Some command have an exclusive form, operating on all variables, except those named. This type of operation is represented in the matrix as a special code against the excluded variable (X for exclusive kill, Y for exclusive new). In addition, a K or N is marked against the pseudo variable '*' to indicate that all other variables could, potentially, be newed or killed by the process.

The Lock command performs an implicit unlock of all other locks (unless it is an incremental lock). The implicit unlock is not shown in the matrix. This characteristic of the Lock command should be taken into account when using the matrix to investigate locking.

The \$get command does not have a unique code within the matrix. Instead it is represented as a combination of D and R (\$data and read).

When a variable is marked as being output (code O) this may mean that only part of the contents of that variable are output. The contents may also be transformed as part of an expression before being output. Figure 12.3 shows some examples of WRITE command arguments and the resulting contents of the matrix for each case.

Variable	1	2	3	4	5				
v1	RO	RO	RO		RO				
v1(-,-)				RO					
v2	RO		RO	RO	RO				
x	R	R		R	R				
y	R	R		R	R				

Figure 12.3 WRITE Command Examples

Screen Operation

Analyze Process Data Structure Global Record Usage Matrix Exit Help Process/Data Matrix											
Repository System Group Transaction Program Exit Help System Process/Data Matrix											
Up Down Left Right Origin Print Screen Cursor Tab Exit Help Navigate around the matrix											
T r a n s a c t i o n s											
System SALES	SMA	SMB	SMC	SMD	SME	SMF	SNA	SNB			
Global Reference											
^df(-)		K	RT		RT		RT				
^df(-,-)			R		T		RT				
^df(-,-,-)			T		RT		T				
^df(-,-,-,-)			T		RTD		RST				
^en			S								
^en(-)		KLU						LU			
^en(-,-)	RSTD	RST	RT					RSD			
^en1	T	RST	T					RSD			
^en1(-)		K						RT			
^en1(-,-)		RSK						RT			
^fn(0)		RD					RSLU				
^fn(0,-)		RT	RT				RSTD				
^fn(1,-)		RS					RSTD				
^fn(1,-,-)		RST									
^sm(-)	RSDK				RT						
^sm(-,A,-)	RT				RT						
^sm(-,B,-)	RST				RT						

Figure 12.4 System Level Process/Data Matrix

o **Up Down Left Right**

When a matrix is larger than the screen, the commands Up, Down, Left and Right enable different parts of the matrix to be examined, by moving the matrix in the selected direction. The effect of these commands depends upon the display mode that has been selected. See screen, cursor and tab commands below for more details.

On most terminals, the arrow keys may be used to perform the same function as Up, Down, Left and Right.

o **Origin**

The Origin command returns the display to the top-left hand corner of the matrix.

o **Print**

The Print command prompts for an output device, which may be a printer or an ASCII format sequential file (subject to Operating System support), and prints the complete Process/Data matrix.

o **Screen** (Default)

The screen option selects the screen paging mode. In this mode when the arrow keys are pressed the matrix display will pan in the direction indicated.

o **Cursor**

If the cursor mode is selected highlighted vertical and horizontal cursor bars are displayed on the matrix. Using the arrow keys will move the cursor bars in the direction indicated by one row or column at a time.

o **Tab**

If the tab mode is selected then cursor bars will be displayed, as for the cursor mode. However, when the arrow keys are used to move the cursor bars instead of moving one column or row at a time the cursor bar will tab to the next cell that contains an entry. This enables the entries in sparse matrices to be viewed quickly and accurately.

Record Content Diagrams

Contents

Diagram Notation	104
Screen Operation	105

Record structure diagrams are used to analyze the field structure of records. Each global node represents a record that may contain data. In many M applications the data in a record is stored in variable length fields separated by a delimiter character, such as an asterisk (*), a backslash (\), a tilde (~) or an up-arrow (^). Each item of data is unpacked from a global node using the \$PIECE function.

The Record Structure Diagrams show how the contents of a global node (record) is unpacked into its component parts. This information can be used to deduce the field structure of the global node (record) and, if meaningful variable names have been used, to identify the contents of each field.

The Record Structure Diagrams are represented as a hierarchy with the source variable at the top of the diagram. Each diagram shows all variables that have been derived from the source variable. This process is then repeated for each derived variable until all the derivations from the original source variable have been traced.

```

Database:
    ^CUST(10001)=John Smith\1217 Chestnut Street\Philadelphia\PA\10567\19501104
    ^CUST(10002)=John Doe\2154 Elm Street\Oakland\CA\10345\19640814

Program:
    .
    .
    S CU=^CUST(CUNO)
    S NAME=$P(CU,"\",1),FORENAME=$P(NAME," ",1),SURNAME=$P(NAME," ",2),INTL=$E(FORENAME,1)
    S ADDR1=$P(CU,"\",2),ADDR2=$P(CU,"\",3),STATE=$P(CU,"\",4),ZIP=$P(CU,"\",5)
    .
    .
    S DOB=$P(CU,"\",6),Y=$E(DOB,1,4),M=$E(DOB,5,6),D=$E(DOB,7,8)
    .
    .
    S FNAME=SNAME
    .
    .

```

Figure 13.1 Example Database and Code

An example illustrates this. Analysis of the code shown in figure 13.1 would produce a diagram such as that in figure 13.2. First of all the variable CU is derived from ^CUST(CUNO). Then six fields within the variable CU are un-pieced. The position of these items within the record are shown on the diagram and identify the location of each data item within the database.

Following on from this the code analyzer has also identified that the name variable is broken down into forename and surname. And that the person's initial is derived from the forename. The question mark (?) against the derivation of INTL indicates that there is not a direct equivalence between the contents of FORENAME and INTL. Contrast this with the derivation of FNAME from SURNAME which is an exact equivalent.

Typically, database accesses are scattered throughout a system. RE/m is able to gather together information about data derivations from wherever it occurs within the selected set of processes to create a composite diagram of all data that is derived from a single global node.

o **Limitations:**

Unlike the other analysis techniques within RE/m, the record structure analysis does not produce definitive system documentation. The nature of the analysis methods used means that the resultant diagrams can only be considered as first-cut indications of the data held within global nodes. In all cases further manual interpretation is required to completely document the field contents of a global node.

There are two specific limitations that the user should be aware of:

Destructive Interference can occur when the same intermediate variable (eg X) is used to unpack several different global nodes within the same, or connected, pieces of code. In this situation, RE/m is unable to determine where the intermediate variable originated from when it is subsequently un-pieced. In order to avoid presenting inaccurate information this type of ambiguity is disregarded by RE/m. If this effect occurs, or is known to exist within the code being analyzed, then the resultant record structure diagrams will only be a sub-set of the actual record. In some cases this can result in no information at all.

Noise. Because of the nature of the way that temporary and scratch variables are often used within M programs the Record Structure Diagram will often show derived variables that are only used as transient holders of data. The derivation analyzer cannot discriminate these from more significant variables and therefore shows them along with the more meaningful information. These transient variables are called noise and can be ignored if they have no meaning.

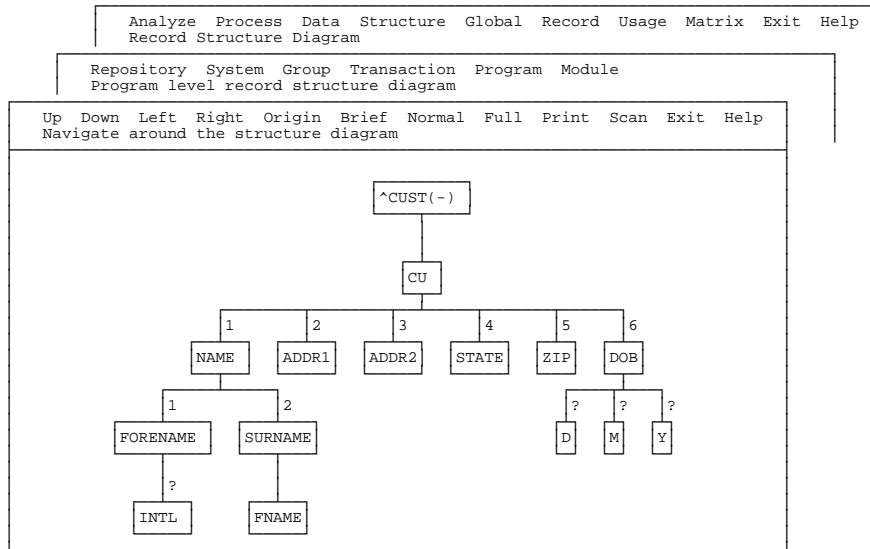


Figure 13.2 Record Structure Diagram

Diagram Notation

The derivation of data is shown as a hierarchy with each variable being represented by a box. The branches in the hierarchy represent the derivation of one variable from the other. Each variable is derived from the variable immediately above it on the diagram.

- o **Direct Equivalence**

Where a derivation is shown between two variables and the line between them has no label on it, this means that the two variables will be equivalent and will contain the same data. The lower variable in the diagram will have been derived from the upper variable.

- o **Piece**

Where a derivation is shown between two variables with a number against the line that links them, this indicates that the lower variable has been un-pieced from the upper variable and that the number indicates the position of the lower variable within the upper one. Note that the piece delimiter is not shown. This is assumed to be standard throughout a system or deducible from the context in which the derivation occurs.

- o **Question Mark**

Where a question mark appears on a link between two variables this indicates that there is a derivation of the lower variable from the upper variable, but it is not a direct equivalence or a single piece of the upper variable.

Typically, the lower variable could have been derived from the upper variable by any arbitrarily complex expression.

- o **Synonyms**

Where the same piece of a variable has been unpacked into two different variables these are considered to be possible synonyms. This situation will be represented on the diagram by the two variables appearing adjacently and by both being labeled with the same piece number.

Screen Operation

Refer to figure 13.2 for an example of the screen layout and options available with the Record Structure diagrams.

- o **Up Down Left Right**

When a diagram is larger than the screen, the commands Up, Down, Left and Right enable different parts of the diagram to be examined, by moving the diagram in the selected direction. Each movement is approximately one eighth of the size of the screen. If the appropriate key is held down the diagram will move more quickly.

On some terminals, the arrow keys may be used to perform the same function as Up, Down, Left and Right.

- o **Origin**

The Origin command returns the display to the top-left hand corner of the diagram.

- o **Brief**

The brief format shows the same information as the normal format but in a very compact form. It is used when a large diagram is being viewed and it is necessary to see as much of the diagram as possible.

o **Normal**

The normal display format shows each variable in a box with a line linking each one to the variable it was derived from. Lines also link the variable to all variables that are derived from it.

o **Full**

The full format display option shows the same information as the normal display option but it also shows the description of the variable if one exists in the data catalog.

o **Print**

The Print command prompts for an output device, which may be a printer or an ASCII format sequential file (subject to Operating System support), and prints the complete derivation diagram in the format of the currently selected display option.

o **Scan**

The scan option provides a way of searching the diagram for the occurrence of a string. This can be used to find the location of an object by entering its name.

The scan will search for any partial or full match on both the name of the object and its description. The description will be searched even if the brief display option is currently selected.

When an object has been found that matches the search string it is highlighted and if it is not currently displayed the display window is moved so that it can be seen.

Installation and Configuration

Contents

Load Software	108
Implementation Specific Code	109
Set-up RE/m	110
Set-up Device Types	111
Set-up Terminals	114
Set-up Printers	115
Configure RE/m	117
Start RE/m	119
MSM Installation Notes	120
DTM Installation Notes	121
VAX-DSM Installation Notes	122
M/VX and M/SQL Installation Notes	123
GT.M Installation Notes	124

The following procedure should be used to install RE/m for the first time.

If you are upgrading a currently installed version of RE/m do not use this installation procedure. Each new version of RE/m is supplied with release notes that describe how to upgrade from an old version of RE/m to a new version.

If you are installing RE/m for the first time you should consult the implementation specific section of this appendix before following the instructions below.

Load Software

RE/m is supplied as a set of M routines and globals in routine save and global save format. They are supplied on the appropriate media as four files:

REM.RSA	- Standard format routine save of RE/m routines
REM.GSA	- DTM format global save of RE/m globals
REM.GTO	- DSM format global save of RE/m globals
REM.MSM	- MSM format global save of RE/m globals

The three global save files contain the same data in three different formats. You should use the one most appropriate for your system. The files should be loaded into the required destination UCI, namespace or directory using the appropriate routine restore and global restore utilities. Approximately 1 Mbyte of disk space will be required to load the RE/m system.

RE/m consists of the following routines and globals:

Routines:	RE* and re**
Globals:	^re Repository
	^re100 Parser file
	^rehelp Help text
	^remnu Menu file
	^reio IO specifications
	^redev Device types
	^retmp Temporary work file
	^reopt System options
	^reana Code analyzer queue and audit trail

Using an appropriate routine restore utility, restore all routines from the REM.RSA file into the area where you will be using RE/m.

Using an appropriate global restore utility, restore all globals from one of the following files. The file you should use is dependent upon your mumps implementation and the Global Save format that it expects:

M Implementation	Global Restore Utility	File to Restore
VAX-DSM	%GTI	REM.GTO
DTM	%gload	REM.GSA
MSM	%GR	REM.MSM
M/VX or M/SQL	%GTI	REM.GTO
GT.M	MUPIP LOAD /FORMAT=GO	REM.GTO

These files are held on disk one of the distribution kit, or on disk two if you have a two disk distribution kit.

Check box

Implementation Specific Code

Select the correct version of routine re000 for your implementation. Make this the current version by overwriting the default version of re000 with the selected version. In most M implementations this can be carried out as follows:

>ZL re000xxx where xxx is the name of your M implementation (see list below).

>ZS re000

M Implementation	Routine to use
PDP DSM-11	re000dsm
VAX-DSM	re000dvx
DTM	re000dtm
MSM	re000msm
M/VX, M/SQL or Caché	re000mvx
GT.M	re000gtm

NB If you are using GT.M the above task can be achieved by copying re000gtm.m to re000.m using the host operating system copy command and then compiling re000.m.

Check box

Backups

Once in use the RE/m repository will contain information that is of significant value. It is recommended that procedures be arranged to make regular backups of the volatile globals used by RE/m. They are:

- ^re
- ^re100
- ^reio
- ^redev
- ^reana

Check box

Set-up RE/m

Invoke the RE/m set-up program by typing:

- >d ^resetup
- or
- >D ^RESETUP

You will be presented with the following screen:

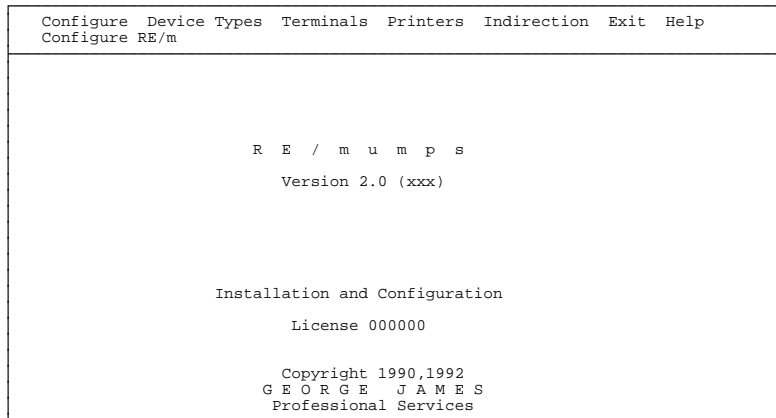


Figure A.1 RESETUP front screen

Check box

Set-up Device Types

You may need to set up the device characteristics of terminals and printers that you plan to use with RE/m. The Device Types menu option in RESETUP enables you to do this. A number of standard device types are pre-defined, as follows:

DEFAULT	Default device type (pre-set as for ANSI)
IBMPC	IBM Compatible PC display (DataTree Console)
VDU	Basic character terminal
ANSI	ANSI standard terminal
LP	Basic character printer
HPL	HP Laser printer (landscape)
HPP	HP Laser printer (portrait)
LN03	DEC Laser printer
MSMCON	MSM console device

If you plan to use devices that are not in the list above then you will need to create an entry in the device type table describing the attributes of that device.

Configure	Device Types	Terminals	Printers	Indirection	Exit	Help
Maintain Device Types						
Device Type	[ANSI]	Printer or Terminal (P/T)		[T]		
Page Width (chars)	[80]	Page length (lines)		[24]		
Enter the ASCII code for the following graphic characters:						
Horizontal bar	[113]	Vertical bar	[120]			
Top left corner	[108]	Top right corner	[107]			
Bottom left corner	[109]	Bottom right corner	[106]			
Inverted T	[118]	T junction	[119]			
Left T	[116]	Right T	[117]			
Cross	[110]	Down arrow	[96]			
Set graphics mode	[14]	Set text mode	[15]			
Initialization	[*27,41,48					
Termination	[

Figure A.2 Device type maintenance

- o **Device Type**
Enter a code that will identify the device type. The system will not permit you to modify device types that are supplied as a standard part of the package. You may however create as many new ones as you need.

o **Printer or Terminal**

Enter P if the device you are defining is a print device. Enter T if it is a terminal or PC screen.

o **Page Width**

Enter the width of the page (or screen) in characters. Where relevant the system will format the screen or page to fit within this width. NB most screen displays and reports assume a minimum width of 80 characters.

o **Page Length**

Enter the length of the page (or screen) in lines. Where relevant the system will format the screen or page to fit within this length. NB most screen displays assume a minimum length of 24 lines.

o **Graphic character ASCII codes**

The next twelve prompts defined the line and box drawing characters that will be used for this device. You should enter the ASCII code corresponding to the character that should be displayed in each case:

Field	Character	Field	Character
Horizontal bar	-	Vertical bar	
Top left corner	┌	Top right corner	┐
Bottom left corner	└	Bottom right corner	┘
Inverted T	⊥	T junction	⊕
Left T	┆	Right T	┆
Cross	⊕	Down Arrow	v

o **Set graphics mode**

If the device uses an alternative character set for graphics characters then enter the ASCII codes required to switch the device to that character set.

Enter each character in the control string as an ASCII code delimited by commas.

o **Set text mode**

If the device uses an alternative character set for graphics characters then enter the ASCII codes required to switch the device back from the graphics character set to the normal text character set.

Enter each character in the control string as an ASCII code delimited by commas.

o **Initialization**

If the device requires some special initialization string (for example, to select compressed print), then enter the control sequence required. This can be entered in any one of the following formats:

Format	Example
Text string	<code>!r!font 9;exit;</code>
ASCII code sequence *nn,nn,nn,...	<code>*27,41,48</code>
Escape sequence @ = escape	<code>@[101;@10c;@[1m</code>

NB the formats cannot be mixed within the same field.

o **Termination**

If the device requires some special termination string (for example, to reset the printer margins) then enter the control sequence required.

It can be entered in any of the formats described for the initialization string.

Check box

Set-up Terminals

For each terminal that will be used with RE/m create an entry in the terminal file (using the Terminal option in the set-up menu). Each terminal is identified by its \$IO value. If an entry is not defined in the terminal file for a device then RE/m will assume that the device is of the type defined as DEFAULT in the device type table. DEFAULT should be configured so as to support all terminals that will be used with RE/m over a terminal server or network.

- o **Terminal (\$IO)**

Enter the \$IO value that identifies the terminal.

- o **Device Type**

Enter the device type of the terminal as defined the the device type table.

- o **Open parameters**

If the device requires special parameters when it is used (for terminals the device will already be open so these will be interpreted as use parameter) then enter them here. If there are multiple parameters then they will probably need to be in brackets. For examples:

Open parameters [(width=0:escape)]

Open parameters [(0:::262208)]

Configure	Device Types	Terminals	Printers	Indirection	Exit	Help
Maintain	Device Types					
	Terminal (\$IO)	[]		
	Device type	[]		
	Open parameters	[]
	Close parameters	[]

Figure A.3 Terminal maintenance

o Close parameters

If the device requires special close parameters on termination then these may be entered here. Multiple parameters may need to be enclosed in brackets.

Check box

Set-up Printers

For each printer that will be used with RE/m create an entry in the printer file. (using the Printer option in the set-up menu). Each printer is identified by a logical name which need not be the same as its \$IO value. RE/m will not be able to use any printer that does not have an entry in this file.

Configure	Device Types	Terminals	Printers	Indirection	Exit	Help
Maintain printers						
Printer name	[LP0]				
Device ID (\$IO)	[30]				
Device type	[LP]				
Open parameters	[(0:::262208)]
Close parameters	[]

Figure A.4 Printer maintenance

o Printer Name

Enter the name by which the printer (or output file) is to be known.

o **Device ID (\$IO)**

Enter the address of the printer or output device. This may be a file name or a device number or a device mnemonic depending upon the operating system and M implementation that you are using.

If you enter "???" in this field then the device ID can be entered at print time following the device name. For example, if a device called FILE is set up with a device ID of ??? then at any Output Device prompt the actual device ID to be used can be input following the device name.

```
Output Device [FILE MYFILE.TXT ]
```

o **Device Type**

Enter the device type of the printer as defined the the device type table.

o **Open parameters**

If the device requires special parameters when it used then enter them here. If there are multiple parameters then they will probably need to be in brackets. For examples:

```
Open parameters [(width=0:escape) ]
```

```
Open parameters [(0::::262208) ]
```

If three question marks are entered at any point in the open parameters field then, in a similar way to the Device ID, a value can be supplied at print time which will be substituted for the three question marks. For example, if the open parameters contained the following:

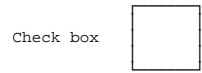
```
Open parameters [(file="???":mode="W") ]
```

Then at the Output Device field the file name can be specified at print time:

```
Output Device [FILE MYFILE.TXT ]
```

o **Close parameters**

If the device requires special parameters on closing then they may be entered here. For example, this can be used to submit print files to a print queue once the report has been created.



Configure RE/m

Select the Configure option from the RESETUP main menu. The screen in figure A.5 will be presented.

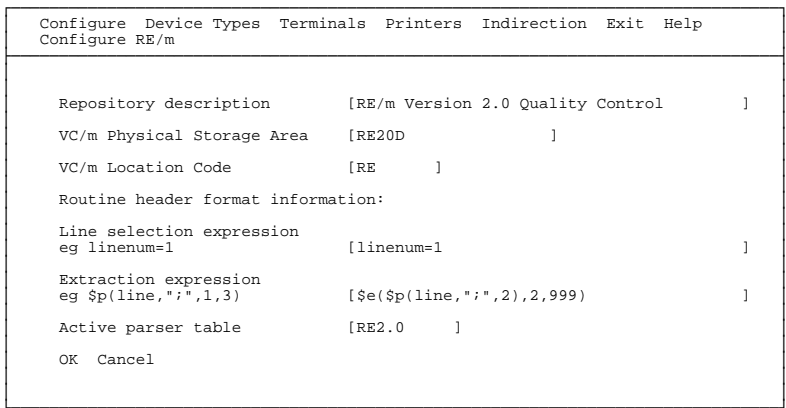


Figure A.5 RE/m Configuration Screen

o **Repository description**

This is a free text field that describes your repository. It will appear on the start up screen of RE/m. If you have several repositories then this can help to ensure that you know which one you are using at any particular time.

NB if you type Z at any menu the RE/m start up screen will be re-displayed.

o **VC/m Physical Storage Area and VC/m Location Code**

If you have VC/m installed then you will be prompted for these items. Enter the physical location code and the logical location codes that will be used to represent the RE/m repository. If they do not already exist then they will be created by this process.

o **Line selection expression**

Enter an expression that will identify the source code line from which the code analyzer should extract the routine description. In most cases this should be line one which can be specified using the expression `linenum=1`. The following variables are available that can be tested:

<code>linenum</code>	The current line number in the routine
<code>line</code>	The contents of the current line
<code>routine</code>	The name of the routine being analyzed

o **Extraction expression**

Enter an expression that will extract the routine description from the selected line of each routine. In most cases this should be some form of \$piece expression. The same variables are available as for the Line selection expression above.

o **Active Parser Table**

Version 2.0 of RE/m provides two alternative parser tables. This prompt enables you to specify which parser table you want to use. If you create your own custom tables then one of these may be specified at this point instead.

RE2.0 is the normal table which will parse any M code conforming to the ANSI 1990 standard plus a number of very common extensions (eg `zallocate`, `zdeallocate` and the two argument `$order`). It also parses all Z commands and functions and other allowed ways of extending the language (open, use, and close command arguments, view and `$view` arguments, break command arguments, write /device functions, etc) in a pass all mode. Any syntax that is a valid form of extension will not be report as an error if this parser table is used.

The REansi90 table does not permit any syntax that does not conform to the 1990 ANSI standard. This table should be used if you are particularly interested in checking that your code conforms to the standard, for portability or other reasons.

Check box

Start RE/m

Check that the installation procedure has been performed correctly by starting RE/m. This is done by typing the following command at the > prompt:

```
>d ^re
```

or

```
>D ^RE
```

If the system has been correctly installed then the RE/m main menu and copyright notice should be displayed.

RE/m is now installed and configured for use. Consult chapter 2 of the Reference manual for full details of how to use RE/m.

Check box

MSM Installation Notes

These notes describe various MSM specific implementation problems and give guidelines on how to configure RE/m to run under MSM.

o Installation Utilities

The following MSM utility programs may be required during the installation and upgrading of RE/m:

```
Routine Save:    %RS
Routine Restore: %RR
Global Save:     %GS
Global Restore:  %GR
```

o Console Device Configuration

If you intend to use RE/m from the PC Console then it is necessary to configure terminal device 1 in the RE/m set-up program to operate correctly with MSM.

Invoke the RE/m set-up program (`d ^resetup`). Select the Terminal maintenance option and add the following record:

```
Terminal ($IO) [1 ]
Device type    [MSMCON ]
Open parameters [(0:::262208) ]
```

The open parameters have the following effects:

```
par1=0        No right margin
par5=262208   Enable 8 bit characters
               Enable Escape processing
```

If any of these parameters have already been set up in the SYSGEN for device 1 then they do not need to be set above. If device 1 is configured in any other way in the SYSGEN then it may be necessary to alter they above parameters to compensate for this.

DTM Installation Notes

These notes describe various DTM specific implementation problems and give guidelines on how to configure RE/m to run under DTM.

o Installation Utilities

The following DTM utility programs may be required during the installation and upgrading of RE/m:

```
Routine Save:      %rsave
Routine Restore:   %rload (DSM-11 format)
Global Save:       %gsave
Global Restore:    %gload
```

o Console Device Configuration

If you intend to use RE/m from the PC Console then it is necessary to configure terminal device 1 in the RE/m set-up program to operate correctly with DTM.

Invoke the RE/m set-up program (`d ^resetup`). Select the Terminal maintenance option and add the following record:

```
Terminal ($IO)    [1    ]
Device type       [IBMPC ]
Open parameters   [                ]
```

o Color Configuration

It is possible to configure DTM to run RE/m in color on the console device. This can be done by mapping the normal ANSI Video Attributes to various colors using the DTM routine `%config`. The following Video Attributes are used by RE/m and some color mappings are suggested:

Video Attribute	Suggested Color Mapping	
Normal	127 (White on Grey)	63 (White on Cyan)
Bold	113 (Blue on Grey)	49 (Blue on Cyan)
Reverse	79 (White on Red)	79 (White on Red)
Reverse Bold	31 (White on Blue)	31 (White on Blue)

VAX-DSM Installation Notes

These notes describe various VAX-DSM specific implementation problems and give guidelines on how to configure RE/m to-run under DSM on VAX systems.

o Installation Utilities

The following VAX-DSM utility programs may be required during the installation and upgrading of RE/m:

Routine Save:	%RS
Routine Restore:	%RR
Global Save:	%GTO
Global Restore:	%GTI

M/VX and M/SQL Installation Notes

These notes describe various M/VX and M/SQL specific implementation problems and give guidelines on how to configure RE/m to run under Intersystems implementations.

o Installation Utilities

The following M/VX or M/SQL utility programs may be required during the installation and upgrading of RE/m:

Routine Save:	%RO
Routine Restore:	%RI
Global Save:	%GTO
Global Restore:	%GTI

GT.M Installation Notes

These notes describe various GT.M specific implementation problems and give guidelines on how to configure RE/m to run under GT.M.

o **Installation Utilities**

The following GT.M utility programs may be required during the installation and upgrading of RE/m:

Routine Save:	%RO
Routine Restore:	MUPIP CONVERT/FORMAT=RO
Global Save:	MUPIP EXTRACT/FORMAT=GO
Global Restore:	MUPIP LOAD/FORMAT=GO

o **Case Sensitivity**

GT.M for VAX/VMS systems is not sensitive to the case of routine names. RE/m is shipped as lower case routines with some upper case equivalents for the two main entry points. As RE/m is loaded the upper case routines will get overwritten by the lower-case equivalents. Because the RE/m upper case routines are functionally the same as their lower case equivalents this does not cause a problem.