Open M/SQL Developer Guide

Version: Open M/SQL F.6, F.7

Revision Date: April 25, 1996

Print History

Creation Date: July, 1992

Revision Dates: March, 1993
September, 1995

Open M/SQL Developer Guide

For Support questions about any InterSystems products, contact the InterSystems Worldwide Support Center:

U.S.:
Tel: +1 617 621-0700
Fax: +1 617 494-1631
Europe:
Tel: +44 753 830-077
Fax: +44 753 861-311
Internet — support@intersys.com

Preface

Open M/SQL is an integrated environment for developing and running database applications. It includes an advanced relational database management system, an application and report generator, and a procedural programming language.

Open M/SQL combines two ANSI-Standard languages — M, the only procedural programming language designed specifically for database applications, and SQL (Structured Query Language), the most widely used relational query language.

Open M/SQL provides two program development environments:

- The M environment, in which you can execute system management commands, and create, modify, and execute M routines.
- The relational environment, in which you define the database in relational terms and use the application generator tools to create, modify, and execute advanced relational database applications.

Open M/SQL's relational environment is a complete RDBMS (Relational Database Management System)—it includes a development environment for creating advanced relational database applications, a management system for maintaining them, and a run-time environment for executing them.

Open M/SQL's relational environment has the following components:

- **Relational Data Dictionary** — environment where you define tables to represent the logical structure of a relational database, or map the physical structure of an existing database to relational tables. These tables act as the foundation on which Open M/SQL relational database applications are built.
- **Form Generator** — an application generation tool used to create ready-to-run data screens through which end users can add, retrieve, edit, and delete database information.

- **Menu Generator** — an application generation tool used to create pop-up menus and menu bars which unite the various components of an application in a logical and visually sophisticated manner and structure an application by defining how it is organized and how it is presented to end users.
- **M/PACT** (Report Generator) — a report-writing tool used to create and run sophisticated end-user data reporting applications that interact with the Open M/SQL Relational Data Dictionary.
- **Query Generator —** environment in which you can create and run database queries on an ad-hoc basis.
- **Relational Server** — an optional component of Open M/SQL that allows non-M applications to access and modify your Open M/SQL relational database.

The *Open M/SQL Developer Guide* describes all aspects of Open M/SQL program development that are independent of considerations specific to hardware platform, operating system, and host M system.

This document provides an overview of the Open M/SQL program development environment (including both the M environment and the relational environment) and presents various strategies for developing applications.

More specifically, it contains information on the following topics:

- Development of macro source routines
- Use of the Full Screen Editor
- Use of routine management utilities
- Use of the Open M/SQL developer utilities
- Overview of Open M/SQL's implementation of SQL
- Various methods for using SQL to query an Open M/SQL relational database

Audience

This document is intended for use by programmers who are using InterSystems' Open M/SQL Developer product to create Open M/SQL relational database applications.

The term "application programmer" in this document refers to the programmers who create Open M/SQL relational database applications.

The term "user" in this document refers to the end user of an Open M/SQL relational database application.

This document assumes that you have programming experience. In specific, it assumes familiarity with programming in the M language and the SQL language.

Organization of this Guide

This manual is divided into four sections. Section I, *Getting Started*, contains the following two chapters:

- Chapter 1, *Introduction to Open M/SQL*, introduces InterSystems' Open M/SQL and discusses its uses in a 4GL application generator environment. Chapter 1 begins with a discussion of application development strategies, then goes on to discuss the components of Open M/SQL and their various applications.
- Chapter 2, *The Open M/SQL Relational Database*, describes relational concepts and their implementation in SQL as well as the database concept and its implementation in Open M/SQL.

Section II, *Program Development*, contains the following five chapters:

- Chapter 3, *Open M/SQL Program Development*, outlines the range of facilities available for program development in Open M/SQL.
- Chapter 4, *Full Screen Editor*, describes the Open M/SQL Full Screen Editor used in editing macro source routines, intermediate code routines, and include files.
- Chapter 5, *Developing Macro Source Routines*, describes how to create routines at the macro source level, including descriptions of all the macro constructs permitted by Open M/SQL at this level.
- Chapter 6, *Routine Handling and Maintenance*, presents an overview of the Open M/SQL routine environment and describes routine handling and maintenance.
- Chapter 7, *Open M/SQL Routine Management Utilities*, describes the built-in utilities that InterSystems supplies to assist in examining and manipulating routines and include files.

Section III, *SQL Language Implementation*, contains the following three chapters:

- Chapter 8, *Embedded SQL*, describes the two ways of using SQL in an Open M/SQL program: cursor-based and non-cursor-based.
- Chapter 9, *Open M/SQL Implementation of SQL*, describes extensions to ANSI-Standard SQL that InterSystems has incorporated into Open M/SQL.
- Chapter 10, *Query Generation and Processing*, describes the Query Generator and the Query Editor—the two facilities provided by Open M/SQL for the generation and processing of ad hoc SQL queries.

Section IV, *Application Programming*, contains the following chapter:

- Chapter 11, *Programmer Interface to Applications*, presents an overview of the M variables and globals that interact with forms and reports. It also discusses the uses of inserted M code in an application, the conventions for calling forms, reports, queries, and menus and emulating forms, and the contents of applications.
- Chapter 12, *Open M/SQL Developer Utilities*, describes the built-in utilities that InterSystems supplies to assist in examining and manipulating routines and include files.

This guide contains the following appendices:

- Appendix A, *SQL Error Messages*, which lists and describes all message codes (successful completion messages and error messages) that can be returned to the SQLCODE variable.
- Appendix B, *Open M/SQL Reserved Words*, which lists all Open M/SQL reserved words.
- Appendix C, *Open M/SQL Supported Terminal Types*, which lists all terminal types currently supported by Open M/SQL.
- Appendix D, *Keyboard Actions for the Full Screen Editor*, which shows the keyboard mapping for using the Full Screen Editor on keyboards associated with the terminal types supported by Open M/SQL/

A glossary defines the terminology used in this guide.

Other References

Depending on the configuration of your system, you may also wish to refer to the following guides:

- The *Open M/SQL Database Administrator's Guide*, which describes how to configure, manage, and maintain an Open M/SQL environment, including both the application development environment and the application run-time environment. For sites that use the Open M/SQL Relational Server to make Open M/SQL data available to external applications, this guide also describes how to configure, manage, and maintain the Relational Server.
- The *Open M/SQL User Interface Programming Guide*, which describes how to use the application generation tools within Open M/SQL Developer to create a user interface for your relational database. These application generation tools include the **Form Generator** — used to create ready-to-run data screens through which end users can add, retrieve, edit, and delete database information; the **Menu Generator** — used to create pop-up menus and menu bars which unite the various components of an application in a logical and visually sophisticated manner and structure an application by defining how it is organized and how it is presented to users; and the **Application Help Facility** — used to create and deploy a context-sensitive on-line help system for your Open M/SQL relational database application.
- The *Open M/SQL Data Dictionary Guide*, which describes how to define tables to represent the logical structure of a relational database, or how to map the physical structure of an existing database to relational tables. These tables act as the foundation on which Open M/SQL relational database applications are built.
- The *Open M/SQL Relational Client User Guide*, which describes how to access an Open M/SQL relational database (retrieve and modify data) from various third-party application development tools.
- The *Open M/SQL Server Programming Guide*, which describes how to access an Open M/SQL relational database from applications created using the C or C++ programming language.
- The *M/PACT Guide*, which describes how to create and run sophisticated end-user data reporting applications that interact with the Open M/SQL Relational Data Dictionary.
- An tutorial entitled *Open M/SQL: A Gentle Introduction*, which uses a project management demonstration application to show how to develop and use an Open M/SQL application.
- The *Open M/SQL M Programming Guide*, which describes ISM (InterSystems' original implementation of the M programming language), including global database concepts, the development of intermediate code routines, and the use of ISM utilities.

- The appropriate system guide for your Open M system platform, which describes the system-specific elements of using Open M on your particular computer and operating system.

Typographic Conventions Used in this Guide

This guide observes the following typographic conventions:

Convention Description	Example
Body text appears in Times Roman type.	Open M/SQL offers a variety of strategies for developing an application.
Computer generated information appears in Courier type.	`Linking/Syntax Checking...DONE`
Computer generated required fields appear in Courier Bold type.	**`Update New M Database Defaults?`**
User input appears in Courier Bold Italic type.	***`Yes`***
Keystrokes appear in uppercase and enclosed within angle brackets.	<RETURN>
Simultaneous keystrokes appear hyphenated, in uppercase, and enclosed within angle brackets.	<CTRL-Z>
Command syntax appears indented in Courier type with variable information in Courier Italic type and enclosed within angle brackets.	`> do setaid^%msql("<UserName>")`
Examples of user-typed commands appear indented in Courier Bold Italic type.	***`> do setaid^%msql("Zeus")`***
Procedure titles appear in bold Helvetica type and are designated by an underlined sidehead prefix.	**Procedure** **To access the System Configuration window:**
Note, Caution, Warning, and Hint statements appear in Helvetica type with an appropriate sidehead.	**Note** Turning DBMS security OFF is a useful way to guarantee access to current applications by users who will need them while you are in the process of implementing a privilege system.
Titles of other guides and other chapters within this guide appear in Times Italic type.	*Open M/SQL User Interface Programming Guide*

Table of Contents

Preface

Audience iv
Organization of this Guide v
Other References vii
Typographic Conventions Used in this Guide ix

Part I—Getting Started

1 Introduction to Open M/SQL

What Is Open M/SQL? 1-2
What Is Open M/SQL Developer? 1-3
Application Development Strategies 1-4
- Automated Program Generation 1-4
- Procedural Programming in ANSI M 1-6
- Mixed Environments 1-7

Overview of the System Environment 1-8
- Hardware and Operating System Environments 1-8
- Memory Environment 1-9
- Global Database 1-9
- ANSI-Standard M Language Processor 1-9
- ANSI-Standard SQL Language Processor 1-9
- Relational Data Dictionary 1-10
- Program Development Environment 1-10
- The Form Generator and M/PACT 1-12
- Menu Generator 1-12
- Query Optimizer 1-12
- M/NET Networking 1-13

DT Network 1-13
Open M/SQL Runs on Top of Any M Implementation 1-14
Integration of Two ANSI Standards 1-15
Embedded SQL 1-15
Inserted Code for Data Dictionary, Forms, and Reports. 1-15
Accessing the Global Database 1-16
SQL - Relational Database Access 1-16
M Global References 1-16
Open M/SQL Program Structure 1-17
Macro Source Routines 1-17
Intermediate Code Routines 1-17
Object Code Routines 1-18
Open M/SQL Routine Utilities and Editors 1-18
Distributed Data Processing 1-19

2 The Open M/SQL Relational Database

Open M/SQL Implements Relations as Tables 2-2
Open M/SQL Tables Follow The First Normal Form 2-2
Open M/SQL Supports Two Types of Tables 2-3
Open M/SQL Implements Schemas as M Directories. 2-4
Open M/SQL Database Structure for Non-ISM Implementations of M. 2-4
The Open M/SQL Relational Data Dictionary 2-5
Accessing Data in an Open M/SQL Relational Database 2-6
Using SQL to Query the Database 2-6
Cartesian Product 2-7
Project 2-9
Restrict 2-10
Joins 2-11
InterSystems' Extensions to the Relational Model 2-14
One-Way Outer Joins 2-14
Implicit Joins. 2-15
Multi-Line Fields 2-21

Part II—Program Development

3 Open M/SQL Program Development

Programming Methods 3-2
The Full Screen Editor. 3-2
Intermediate Code Routines 3-3
Routine Line Editor. 3-3

Routine Management Utilities 3-3
Developer Utilities 3-4
Programmer Interface to Applications 3-4

4 Full Screen Editor

Overview of the Full Screen Editor 4-2
Full Screen Editor Features 4-2
Routine Types for Editing 4-2
Invoking the Full Screen Editor 4-3
Loading Existing Routines. 4-6
Creating New Routines 4-7
Preventing Overwrites 4-7
Full Screen Editor Screen Display 4-8
Navigating the Full Screen Editor Menu System. 4-10
Primary Menu 4-10
Buffers Menu. 4-11
Windows Menu 4-11
Mark Menu 4-12
Other Menu 4-12
Editing Operations 4-13
Moving the Cursor. 4-13
Inserting Text. 4-14
DeletingText 4-14
Cutting and Pasting Text 4-15
Editing Multiple Copies of a Routine 4-16
Displaying Multiple Buffers 4-18
Setting a Mark in Your Current Buffer 4-19
Searching For Text Strings 4-20
ReplacingText Strings 4-20
Using Control Key Commands for Quicker Editing 4-20
Getting Help 4-22
Exiting the Full Screen Editor 4-23
Automatic Syntax Checking 4-23
Automatic Date and Time Stamps. 4-24

5 Developing Macro Source Routines

Creating Macro Source Routines 5-2
Compiling Macro Source Routines 5-2
Macro Source Routines and Include Files. 5-3
Macro Source Routines Are Portable Across M Implementations 5-3

The Open M/SQL Macro Preprocessor 5-4
Macro Preprocessor Commands 5-4
Macro Preprocessor Functions 5-5
Macro References 5-5
Summary of Macro Preprocessor Commands 5-7
#define and #undef 5-7
Nested Expansion 5-9
#ifdef, #ifundef, #if, #else, #elseif, and #endif 5-11
#include 5-13
Indicating Comment Lines 5-15
Summary of Macro Preprocessor Functions 5-16
&sql(...) 5-16
##vendor 5-17

6 Routine Handling and Maintenance

Routine Environment 6-2
Writing Routines 6-2
Converting Intermediate Code to Macro Source Code 6-2
Routine Names, Extensions, and Version Numbers 6-3
Routine Names Must Be Unique 6-3
Case Sensitivity 6-3
Routine Extensions 6-3
Version Numbers 6-5
Using Wildcard Symbols to Specify Routines 6-6
Wildcards for Routine Names 6-6
Wildcards for Extensions 6-7
Wildcards for Version Numbers 6-7
Referencing Routines in Other Directories 6-8
Restrictions on Using Remote Directory Syntax 6-9
Routine Sets 6-10
Creating a Routine Set 6-10
Using a Routine Set 6-10
Compiling Routines 6-11
Backing Up Routines 6-12
How Backups are Shuffled and Renumbered 6-12
The Full Screen Editor Generates Backups When You Save 6-13
Restoring a Backup Version to the Current Version 6-13
Deleting Routines 6-14
%urdel 6-14
%urpurge 6-14
Routine Copying and Compiling Synchronization 6-15

7 Open M/SQL Routine Management Utilities

Summary of Routine Management Utilities 7-2
Accessing the Routine Management Utilities 7-3
Calling the Routine Utilities Directly from M 7-3
Accessing the Routine Utilities from within Open M/SQL 7-3
%urprint 7-7
Selecting an Output Device 7-7
%urload 7-9
Routine Input Options 7-9
Compile Macro Source Routines 7-10
%urdir 7-11
%urchange 7-15
%urcomp 7-17
%urcopy 7-19
%urfind 7-23
%urfand 7-25
%urdel 7-27
%urverma 7-28
%urpurge 7-29
%urset 7-30
The Global ^mtemp 7-30
Parameters of %urset 7-31

Part III—SQL Language Implementation

8 Embedded SQL

Preprocessor Syntax Delimits Embedded SQL 8-2
Open M/SQL Supports Two Kinds of Embedded SQL 8-2
Non-Cursor-Based SQL 8-2
Cursor-Based SQL 8-4
Referencing Macros in Embedded SQL 8-7
Internal and External Values 8-8
Multi-Line Values 8-8
Reserved Tag and Variable Names 8-9
Portability 8-9
Detailed Example 8-10

9 Open M/SQL Implementation of SQL

Summary of Extensions 9-2
Added Keywords and Symbols 9-3
Joins 9-5
One-Way Outer Joins 9-5
Implicit Joins 9-6
Aggregate Extensions 9-11
Aggregates as Query Columns 9-11
%FOREACH 9-12
%AFTERHAVING 9-14
DISTINCT BY 9-14
Duplicate Rows 9-16
Embedded SQL 9-17
Using the INTO Clause 9-18
You May Use INTO in Cursor Declaration 9-18
Using INTO with Arrays 9-19
SELECT INTO Using Arrays 9-19
INSERT and UPDATE INTO Using Arrays 9-21
VALUES Extension for INSERT and UPDATE Queries 9-23
Multi-Line Fields 9-24
Using Multi-Line Fields in SELECT Queries 9-24
Using Multi-Line Fields in INSERT and UPDATE Queries 9-25
Internal and External Values 9-26
%INTERNAL and %EXTERNAL Functions 9-29
M Operators 9-30
Pattern Match Operator Can Test Variables 9-31
Extensions to SQL Operators 9-32
[NOT] IN Operator 9-32
Concatenation Operator 9-32
%STARTSWITH 9-33
LIKE Predicate 9-34
Special Pattern Matching Characters 9-34
Specifying a Host Variable as the Search Pattern 9-35
ESCAPE Qualifier 9-35
Error Handling 9-36
Collation Sequence 9-37
EXACT 9-37
ALPHAUP 9-38
UPPER 9-38
Plus, Minus, and Space 9-39

Field Collation Sequence 9-39
Collation Sequence and ORDER BY 9-40
Collation Sequence and Comparisons 9-40
%ALPHAUP, %UPPER, and %EXACT 9-41
Changing the Default Collation Sequence 9-42
%NOCHECK 9-44
SQL Transaction Processing 9-45
Privilege Operators 9-47
GRANT 9-47
REVOKE 9-49
%CHECKPRIV Keyword 9-50
SQLCODE Values 9-51
Using Subqueries 9-52
Using a Subquery in a WHERE Clause 9-52
Using a Subquery in a FROM clause 9-54
Open M/SQL Subquery Extensions 9-55
Query-Based Views 9-57
CREATE VIEW 9-57
ALTER VIEW 9-57
DROP VIEW 9-58
Restrictions on Defining Query-Based Views 9-58
You May Name Query-Based Views in FROM Clause 9-58

10 Query Generation and Processing

Facilities for Creating SQL Queries 10-2
Accessing the Query Generation Facilities 10-2
Using the Query Generator 10-5
Example 10-9
Query Definition Menu Bar 10-11
Using the Interactive Query Editor 10-17
Query Types 10-17
Creating a Query in the Interactive Query Editor 10-18
Editing Commands 10-23
Query Editor Horizontal Options Menu 10-23
Running Queries 10-29
Privileges Required to Run Queries 10-29
Running a Query From its Definition Environment 10-29
Using the Run Existing Queries Utility 10-30
Running a Query from M Code 10-31
Selecting an Output Device 10-32

ASCII-Delimited Output for Queries 10-35
Contents of an ASCII-Delimited File 10-35
Queries Support Dual Output Formats 10-35
Copying Queries 10-36
List Queries Report 10-39
Detailed Query Listing Report. 10-40

Part IV—Application Programming

11 Programmer Interface to Applications

Open M/SQL Variables 11-2
List of Open M/SQL Percent (%) Variables. 11-2
Entry Points to the %msql Routine 11-8
Open M/SQL Globals 11-9
Object Definition Globals 11-9
Open M/SQL Percent (%) Globals 11-12
Inserting Code into Open M/SQL Applications 11-13
Inserted Code Can Reference Fields. 11-14
Inserted Code Can Reference Variables and Globals 11-15
Open M/SQL Performs Syntax Checking on Inserted M Code 11-15
Triggers 11-16
Override Queries for Lookups 11-22
Computed Fields 11-22
Internal/External Conversion Code. 11-23
Additional Validation Code 11-24
Required-Maybe Fields. 11-24
Map Subscripts and Pieces 11-25
NEXT Subroutine 11-25
Conditional Map 11-26
Calling Open M/SQL Objects from M Programs 11-27
Calling Forms 11-27
Calling Reports 11-30
Calling Queries 11-31
Calling Old-Style Menus 11-32
Calling Menu Objects 11-32
Emulating Form Behavior from M Programs 11-34
Displaying Help Text in a Help Text Box 11-34
Writing Message Text. 11-35
Reading Fields. 11-36
Cleaning Up Windows 11-38

Establishing Authorization ID from Programmer Mode 11-39
Intermixing Open M/SQL Objects with User-Defined M Routines 11-40
pushvars^%msqlutl 11-40
popvars^%msqlutl 11-41
$$msqlvars^%msqlutl 11-42
Contents of Applications 11-43
Routine Names. 11-43
Contents of Base Table Routines. 11-44
Contents of Form Routines 11-46
Contents of Menu Object Routines 11-46
Contents of Old-Style Menu Routines. 11-47
Contents of Report Routines 11-47
Contents of Query Routines. 11-47

12 Open M/SQL Developer Utilities

Accessing the Developer Utilities Menu 12-2
Using the Object Compile Driver Utility 12-5
Setting Compilation Option Defaults 12-8
Defining the Contents of a Compilation Configuration. 12-12
Compiling the Configuration 12-20
Viewing the Results of the Last Compilation 12-21
Compiling a Compilation Configuration from M Program Code 12-22
Checking the Integrity of Open M/SQL Objects 12-23
Checks on Base Tables 12-24
Checks on Views 12-24
Checks on Forms 12-25
Checks on Reports 12-25
Checks on Menu Objects 12-26
Checks on Triggers 12-27
Running the Integrity Checker Utility 12-29
Error Fixing Mode 12-34
Searching for Strings in Open M/SQL Objects 12-35
Invoking Macro Source Routine Utilities. 12-40
Querying Objects by Routine Prefix 12-42
National Language Reports 12-44

Appendices

A SQL Error Messages

Successful Completion Messages A-2
Error Messages A-3

B Open M/SQL Reserved Words

C Open M/SQL Supported Terminal Types

Open M/SQL for DSM C-3
Open M/SQL for DTM C-3
Open M/SQL for MSM Environment C-3

D Full Screen Editor Keyboard Actions

Altos D-1
ANSI D-4
CIT-500 . . . . . D-7
Dasher D-11
IBM 3151 ASCII Display Station D-13
IBM PC D-17
QUME D-20
Sun D-23
Televideo 905 . . . . . D-26
DEC VT 100 . . . . . D-29
DEC VT 200 . . . . . D-32
DEC VT 220 . . . . . D-35
WYSE-60 (Native Mode) D-39

Glossary of Terms

Index

PART

I

Getting Started

Chapter 1
Introduction to Open M/SQL

Chapter 2
The Open M/SQL Relational Database

CHAPTER

1

Introduction to Open M/SQL

This chapter introduces InterSystems' Open M/SQL and discusses its uses in a 4GL application generator environment.

Specifically, it covers the following topics:

- What Is Open M/SQL?
- What Is Open M/SQL Developer?
- Application Development Strategies
- Overview of the System Environment
- Integration of Two ANSI Standards
- Accessing the Global Database
- Open M/SQL Program Structure

What Is Open M/SQL?

Open M/SQL is an integrated environment consisting of an advanced relational database management system, application generator, and procedural programming language that offers a variety of strategies for developing and running database applications.

Open M/SQL combines two ANSI Standard programming languages, SQL and M. SQL is the most widely used relational query language. M is a database-oriented procedural language that has been used to develop and run performance-critical online applications at thousands of sites. The merger of these two languages provides a powerful standards-based system that offers extremely high performance.

The "Open" in Open M/SQL refers to its open systems architecture, which allows it to run in a vendor-independent host M environment. Specifically, you can develop and run Open M/SQL applications on top of the following M implementations:

- ISM
- DTM
- DSM
- Micronetics' MSM

All Open M/SQL applications are completely portable between these M systems. This means you can develop an application on one M system, then port it to another M system without making any modifications.

What Is Open M/SQL Developer?

Open M/SQL Developer is InterSystems' development environment for creating Open M/SQL relational applications.

Open M/SQL Developer encompasses the relational Data Dictionary, where you define relational data structures, the Form Generator, where you build interactive data entry screens, and the Menu Generator, where you bring the various components together into a coherent application. It also supports various applications of SQL, including the interactive construction of queries and the embedding of SQL code within M routines. Open M/SQL Developer allows you to produce sophisticated reports, queries, and forms with speed and ease.

The Open M/SQL Data Dictionary describes the elements of an Open M/SQL relational database and as such constitutes the structural foundation of an Open M/SQL application. The Data Dictionary provides an advanced, window-based approach to defining, storing, and retrieving data.

The Open M/SQL Form Generator creates ready-to-run data entry screens to add, delete, retrieve, and edit database information. The generated forms are equipped with data validation, full-screen display handling, powerful function keys, and SQL queries used internally to retrieve and store data.

M/PACT is an Open M/SQL add-on option that allows you to generate highly formatted reports from data served by the relational Data Dictionary.

The Open M/SQL Relational Server is a related product that allows non-M, third party application development tools and C programs on client systems to access your Open M/SQL relational databases on a server system.

Application Development Strategies

Open M/SQL offers a variety of strategies for developing an application: development can take place at a purely automated level; programs can be entirely hand-coded; or, most commonly, applications can be developed in a mixed environment that includes programs created using M, the Open M/SQL procedural programming language, as well as automatically-generated forms, reports, and SQL queries, and other ancillary functions using the application generation tools of Open M/SQL. The following discussion addresses the different strategies for applications development in Open M/SQL.

Automated Program Generation

Automated program generation takes place in five phases:

1. Map the functional specifications for the application out on paper.
2. Define the data structures in the Data Dictionary, along with integrity constraints, table relationships, and triggers.
3. Use the Form Generator to develop data entry and inquiry forms.
4. Use M/PACT to develop reports.
5. Use the Menu Generator to tie the different parts of the application together.

Mapping the Functional Specifications

The first step in developing an application is to organize the logical structure of the data on paper as a series of tables, just as you might design a paper filing system. To do this, consider the functional needs of your application and make a list of all the different pieces of data that you need. The object of the design is to describe the tables that constitute the database and how those tables will interact with one another. For example, one application might have a customer table, an invoice table, an invoice line item table, a parts table, etc. The customer table might then consist of fields for customer name, address, phone, current balance, etc.

Defining the Data Structure in the Data Dictionary

Once you have designed a data structure on paper, you next define that data structure in the Open M/SQL Data Dictionary, along with any integrity constraints on the data. See the *Open M/SQL Data Dictionary Guide* for details about how to create and modify the underlying M global structure for your relational database.

Integrity constraints may be very simple; for example, you can specify a range of valid numbers that can be entered for a field. Other constraints may be more complex; perhaps you want to specify that an invoice cannot be entered into the invoice table without a corresponding customer entry in the customer table. Even

more complex constraints may require the addition of complete SQL queries or procedural M code.

The Data Dictionary will also contain descriptions of the relationships between tables. For example, where rows of an invoice line item table cannot exist without a corresponding row in an invoice table, the line item table should be defined as a **characteristic table** of the invoice table. One field in the invoice table should also be defined to serve as a **designative reference** to rows in the customer table. Open M/SQL uses this information to enforce integrity constraints and produce automatic joins in queries.

You may also define processing triggers in the Data Dictionary. Triggers are sequences of actions defined to automatically occur, or be *triggered*, when certain other events occur. A trigger definition usually consists of an SQL query or M code segment that should be invoked when rows are created, modified, or deleted. For example, if a medical record for a given patient is deleted, you may want to establish a trigger that will automatically delete all of that patient's lab test information from various lab files.

Developing Forms and Reports

Once you have defined the data structure in the Data Dictionary, the next step is to generate forms that enable data entry and inquiry using the Form Generator. See the *Open M/SQL User Interface Programming Guide* for details about how to use the Form Generator.

Forms are sets of windows for entering, retrieving, modifying, and displaying data. The Form Generator creates visually sophisticated windows complete with data validation capabilities, full screen display handling, and powerful function keys. The Form Generator uses internally stored SQL queries to retrieve and store data.

You can generate windows automatically and modify them easily using cut-and-paste editing facilities. For more complex applications, you can enhance the windows by adding SQL queries or M code to the window definition. You can add very simple lines of code, or add entire programs that perform complex calculations or generate their own displays and prompts in cooperation with the Form Generator.

M/PACT generates reports. Reports are defined through a series of questions on the screen; no complex procedural programming is required. See the *Open M/SQL M/PACT Guide* for information about using M/PACT.

SQL queries can be entered interactively through the SQL Query Definition window.

Tying the Application Together with the Menu Generator

The Menu Generator ties the different components of the application together. You may create pop-up menus and menu bars that unite the various components of an application, apply an orderly structure to the application, and offer the user easy access. For details about creating and using the Menu Generator, see the *Open M/SQL User Interface Programming Guide*.

Procedural Programming in ANSI M

It is also possible to develop applications entirely in M, a powerful procedural programming language that affords rapid program development and high performance in a database application environment. M is an ANSI Standard procedural programming language. Other ANSI Standard procedural programming languages include COBOL, C, FORTRAN, PL/1, BASIC, and PASCAL.

Within M programs, data can be accessed in the following ways:

- Directly through M global references that view the database as a collection of arrays;
- Through standard SQL Data Manipulation Language (DML) constructs: insert, update, and delete. Cursors are fully supported. For standard SQL syntax, refer to a commercial textbook or the ANSI X3.135-1989 standard, also known as ISO 9075:1989.
- Through embedded SQL queries that access or edit a single row (provided you have also defined the data structures in the Data Dictionary);
- Through embedded SQL queries that access multiple rows one row at a time using cursors;
- Through embedded SQL queries using UPDATE or DELETE statements that operate on multiple rows.

Direct global references and SQL queries can be used to access the same data.

Programming in a procedural language allows the creation of complex programs that use sophisticated algorithms. However, procedurally generated programs take longer to develop and are more difficult to maintain than applications generated automatically in Open M/SQL. The combined power of M, the Data Dictionary, the Form Generator, M/PACT, and SQL delivers a flexible, productive, and easily maintained environment for application development.

Mixed Environments

While many applications can be developed entirely through automatic generation, and some are so specialized that they must be entirely hand coded, most applications fall somewhere in between. For this, Open M/SQL offers the advantages of a mixed programming strategy. It generally makes sense to use the Form Generator and M/PACT to develop as much of the application as possible, utilizing customization capabilities where necessary. Then, use M to program those portions that have to be written procedurally. Thus, a laboratory application might perform most of its data entry through generated windows but use M for direct input from specialized instruments. The result is an integrated application created and run entirely within Open M/SQL.

Overview of the System Environment

Open M/SQL runs on a variety of platforms and offers a varied systems environment. It also runs in a vendor-independent host M environment, equally supporting the ISM, DTM, DSM, and Micronetics' MSM implementations of M. The following discussion introduces the various elements of the system environment, including:

- Hardware and Software Operating System Environments
- Memory Environment
- Global Database
- ANSI M Processor
- ANSI SQL Processor
- Relational Data Dictionary
- Program Development Environment
- The Form Generator and M/PACT
- Menu Generator
- Query Optimizer
- M/NET Networking
- DT Network
- Open M/SQL Runs on Top of Any M Implementation

Hardware and Operating System Environments

Open M/SQL runs on a variety of hardware platforms and operating systems, ranging from single-user PCs to large UNIX- and VMS-based systems supporting many hundreds of users, including most major supermicro and minicomputer systems from the leading hardware vendors. Operating systems include MS-DOS, Microsoft Windows 3.1, VMS, and Unix.

One of Open M/SQL's most exceptional features is its portability. Although the host operating system may vary, the Open M/SQL environment is consistent across platforms. Applications developed on one system can be moved to another easily and usually with no modification. Frequently, these applications can be ported with no changes.

Memory Environment

Each M process runs as a separate process on the host operating system, except for MS-DOS and Microsoft Windows systems. In addition to a private memory section for each M process, there are two configurable caches, one for database transactions and one for shared re-entrant use of application programs. The global database is paged in a data pool that is shared by all of the processes, and run-time code is shared among processes. This strategy results in a highly optimized, memory-efficient system.

Global Database

The global database is the physical structure that controls the storage of data. The database can be accessed through SQL requests or M global references, as the programmer sees fit. A full set of utilities for database management are also provided.

ANSI-Standard M Language Processor

The M programming language is widely used for database applications, particularly in interactive environments. Although it is best known for offering programmers the ability to develop programs rapidly and change them easily, M is also recognized for providing high performance in interactive transaction-oriented database environments.

M is one of the few languages that is defined by an ANSI Standard, which indicates its importance to a wide audience in the data processing community. Conformity to this standard protects software and hardware investments and facilitates a level of portability uncommon in other programming languages.

M is accepted by ANSI-Standard SQL as a host language for embedded SQL.

ANSI-Standard SQL Language Processor

ANSI-Standard SQL is the most widely used query language for relational database systems. Open M/SQL permits SQL queries to be entered directly through an interactive window, or to be embedded in M routines. SQL can be used not only to retrieve data, but also to insert, update, and delete data. InterSystems' SQL implementation is compatible with ANSI-Standard SQL. InterSystems also provides numerous SQL language extensions to enhance the power of SQL queries.

Relational Data Dictionary

Open M/SQL includes an advanced relational Data Dictionary that allows you to define the database as a group of tables. Extensive capabilities are built into the Data Dictionary to avoid code redundancy. Data Dictionary maps define the connection between the logical and physical storage structure of the tables. Maps can be automatically generated to produce default structured tables. Alternatively, advanced M programmers may wish to define their own maps to conform with the global structures of existing applications or to design a new database structure compliant to application specifications.

Relationships among tables can be described in the Data Dictionary by defining **characteristic** or **designative** relationships between tables. When you define these relationships in the Data Dictionary, joins among tables are "implicit" and occur in queries automatically, freeing the programmer and end-user from the tedious and demanding task of specifying explicit joins in SQL statements.

The Data Dictionary can also be used to enforce integrity constraints that apply to database modifications made using SQL and forms designed using the Form Generator. Processing triggers can be attached to events in database update and form execution that cause related processing to occur automatically. For example, a trigger can specify that when an invoice is deleted, associated line items of the invoice are also deleted. The system thus provides a means to enforce referential integrity and to define application-specific processing in the dictionary, eliminating redundant programming efforts.

A **database** is defined as a functionally-related group of tables that have been explicitly linked in the Data Dictionary. Tables in a single database can be used to create a **view**. Views are considered **virtual tables** because they appear to be tables to the end user, but are not stored as such in the database. A view can be used as a data source in queries and reports. Views also perform a security function by restricting users' access to specified fields in database tables.

Although describing the database in the Data Dictionary is essential to the use of SQL queries, the Form Generator, and M/PACT, it is not required for M programs that access the database only through direct global references.

Program Development Environment

An Open M/SQL application can consist of data structures defined entirely in the Data Dictionary, with forms for data entry and inquiry developed using the Form Generator and reports for data output developed using M/PACT. Alternatively, you can write programs in the ANSI M programming language, which is extended in Open M/SQL to allow the use of macros, include files, and embedded SQL (M is an ANSI-approved host language for SQL). Open M/SQL also provides a full set of utilities for routine development and management.

Hand-Coded Programming in Open M/SQL

For hand-coded programming, Open M/SQL allows you to work at two levels of processing — the **macro source code** level and the **intermediate source code** level. Typically, you create routines a the macro source level, though you may also create routines at the intermediate code level. Macro source code permits the definition of macros and embedded SQL statements using a combination of ANSI Standard M syntax, special macro preprocessor commands, and ANSI Standard SQL. The macro preprocessor phase of the Open M/SQL Compiler converts macro source code into M routines called intermediate code. You can view and edit intermediate code routines using the Full Screen Editor. From intermediate code the Open M/SQL Compiler generates executable code, called **object code**.

Macro source code can make use of **include files**, which contain definitions that are used in the preprocessor phase of compilation to expand macros and determine whether optional lines of code should be included. They can also be used to include a common block of code in several routines, saving the overhead of calls to a common subroutine.

Open M/SQL Interpreted Compiler

Open M/SQL combines the best elements of interpreted and compiled systems. Interpreters are easy to use because they allow program development, testing, and modification in a simple, integrated process. Fully compiled code runs faster than interpreted code but typically involves more work for the programmer.

Open M/SQL provides program debugging facilities at the intermediate code level. These debugging facilities operate like interpreted systems. When an error or breakpoint occurs, the routine is suspended and a message appears indicating exactly where the error or breakpoint occurred. You can examine and modify variables, arrays, and database data, edit the routine, and resume debugging at any location in the program without restarting the testing process.

When you insert a new line into an Open M/SQL intermediate code routine, Open M/SQL's incremental Compiler automatically processes the code, producing efficient object code. The system always executes the object code version of the routine — never the intermediate code version. The Open M/SQL incremental Compiler is totally transparent; you need never issue separate compilation or linkage commands explicitly.

For purposes of security, you can delete the macro source or intermediate code version of any routine and still execute the object code. Furthermore, by simply erasing a line with the ZREMOVE command and inserting a new line in its place with the ZINSERT command, you can edit an object code routine as if the source code were present. Open M/SQL will maintain an audit trail of any new lines.

The Form Generator and M/PACT

The Form Generator allows you to automatically generate data entry/inquiry forms from Data Dictionary table definitions. You can then modify these forms to your exact specifications using a screen editor called QuickForm. The Form Generator also supports the insertion of M code and SQL statements to further enhance the processing scope of your forms.

M/PACT is an add-on option that allows you to generate highly formatted reports based on data from the Data Dictionary. Designed for use by non-programmers as well as programmers, M/PACT enables you to design sophisticated reports without the use of a complex command language. Reports are generated from your answers to questions logically presented in an easy-to-use window-based interface. Though an M/PACT development license is required in order to develop M/PACT reports, an RDBMS Engine license is sufficient to run M/PACT reports.

Menu Generator

Open M/SQL includes a Menu Generator that ties a variety of programs and options together. The Menu Generator can create two types of menus: **menu objects** and **old-style menus**.

Menu objects are definitions that include the list of options for a menu, its screen positioning, and its display attributes. Menu objects can be run as either **menu bars**, which display horizontally across the screen, or **pop-up menus**, which display options vertically on the screen.

Old-style menus can be either **vertical menus**, which display a variety of options in a vertical list, or **horizontal menus**, which display options in a horizontal list across the bottom of the screen. Menu options can include the following: M code, SQL code, routines, forms, windows, reports, queries, and other menus.

Query Optimizer

An unseen but extremely important part of Open M/SQL is the query optimizer. During program compilation and query processing, the query optimizer analyzes hundreds and sometimes thousands of different solutions to an information request, ultimately selecting the best solution. The result is compiled routines whose speed and efficiency guarantee optimal database throughput and response time, based on the application and database design.

M/NET Networking

The database need not reside on a single computer. M/NET is an add-on option that supports both local area and wide area networks within Open M/SQL. M/NET works transparently, i.e. it is invisible to both the programmer and end-user.

M/NET can access remote databases and split a database across multiple computers. M/NET also allows mixing the hardware of different vendors. M/NET features the Distributed Cache Protocol (DCP), which is a high-speed, block-oriented networking protocol that is proprietary to InterSystems. Some of the other networking protocols currently supported by M/NET are:

- Ethernet
- TCP/IP
- DSM-DDP
- OMI
- UDP
- RS-232

DT Network

DT Network is InterSystems' PC network technology for distributing databases over Local- and Wide-Area networks (LANS and WANS). It supports client/server architecture flexibly, in that each networked computer can act as a client, a router, a server, or all three. DT Network supports a variety of database protocols, including:

- Distributed Cache Protocol (DCP)
- ISNET for sharing globals with Open M/SQL platforms
- Open M/Interconnect (OMI), for connectivity to other M databases
- DSM/DDP, InterSystems' implementation of Digital Equipment Corporation's Distributed Database Protocol (DDP)
- Serial Networking using serial lines for connectivity

It is possible to use multiple database protocols on the same machine. These database protocols also operate over a variety of network protocol stacks. For more information, see the *DT-MAX Network Configuration Guide*.

Open M/SQL Runs on Top of Any M Implementation

Open M/SQL runs in an environment that is independent of the underlying M implementation. You can develop and run Open M/SQL applications on any of the following host M systems:

- ISM
- DSM
- DTM
- Micronetics' MSM

No matter what M implementation you are using as the host environment, Open M/SQL supports full application development and run-time capabilities. This allows existing users of these M systems to protect their current software investments while taking full advantage of Open M/SQL's extensive relational database and application development capabilities.

On ISM systems, Open M/SQL stores routines in a global called ^ROUTINE. On all other host M systems, Open M/SQL stores routines in a global called ^mroutine.

On all of the above host M environments, you may create and edit macro source routines using the Open M/SQL Full Screen Editor. You may also use the full suite of Open M/SQL routine management utilities to operate on routines.

Note To comply with all M implementations, InterSystems has renamed the routine management utilities to be implementation-indepenedent. The implementation-independent names of the utilities all begin with "u" and are lowercase. See Chapter 7, *Open M/SQL Routine Management Utilities* for more information about these utilities.

Open M/SQL applications are completely portable across M systems. This means applications can be developed on one M system and ported to other M systems (for continued development or run time purposes) without modification, as long as they are written in ANSI-Standard M code and do not include implementation-specific extensions. This portability is made possible by Open M/SQL's Export/Import facility, which generates programs specific to a supported M system without disturbing the programs on the development system.

Integration of Two ANSI Standards

Open M/SQL integrates two ANSI Standard programming languages: SQL and M. SQL is a standard query language for relational databases. M is a powerful procedural programming language and ANSI-approved host language for SQL. Together they form an integrated environment for applications development.

Embedded SQL

SQL statements that access the database to select, insert, update, or delete rows (records) can be directly embedded within M routines. SQL statements inserted into M routines are called embedded SQL statements. Non-embedded queries are SQL queries entered into a special query template or entered interactively through a query editor; the results of a non-embedded query can be output to the terminal screen or to a printer. If the query is a SELECT query, the result may or may not be a single row.

Cursors

Embedded SQL statements can be cursor-based or non-cursor-based. A cursor-based statement is used to retrieve multiple rows from a table repetitively, one row at a time. A cursor-based query includes a DECLARE statement, an OPEN statement, one or more FETCH statements, and a CLOSE statement.

For example, if a programmer is writing a routine to access all invoices for a particular customer that are older than a given date, his routine must: 1) DECLARE a cursor, 2) issue an OPEN of the specified cursor, 3) perform a FETCH repetitively on that cursor to get the next invoice that meets the query criteria, and finally 4) perform a CLOSE on the cursor when the processing is complete. An UPDATE or DELETE could also be used following a FETCH to edit or delete the row returned by the FETCH.

A non-cursor-based SQL statement embedded in a routine performs a single operation, such as retrieving a single row, inserting a new row into a table, or updating or deleting a specific row. A non-cursor-based SQL statement is appropriate when the operation is intended to act on only one row.

Inserted Code for Data Dictionary, Forms, and Reports

Programmers can also insert M and SQL code into tables, forms, and reports in a variety of places, including triggers, lookup queries, computed field definitions, and conversion and validation codes. This allows the programmer to take full advantage of Open M/SQL's automatic generator capabilities without costing him/her the ability to create highly customized applications.

The powerful syntax and flexible variable and database structures of M makes it, along with SQL, an excellent language for providing this customization.

Accessing the Global Database

The Open M/SQL global database can be accessed via the SQL language or the M language, both together, using SQL code embedded within M code, and independently. In this way existing M applications can incorporate the use of SQL over time; no large conversion is required.

SQL - Relational Database Access

SQL can be used for database queries or to insert, update, or delete rows in a table. SQL queries can be entered interactively and the results displayed on a terminal, or they can be embedded within M routines. Global structures that were built in a non-relational manner by M programs can be described relationally in the Data Dictionary. This is consistent with the relational model, which does not demand a specific physical storage structure or design. This special feature eases the migration to SQL and the application generator features of Open M/SQL for clients whose applications were developed with M.

M Global References

M also allows the use of global references for direct access to and modification of the database. It is possible to make direct global references to global structures, regardless of whether they have been defined in the Data Dictionary. The same data can be referenced by either SQL or global references. The choice of access technique is up to the programmer and database manager. Thus, existing M applications can incorporate the use of SQL over time; no large conversion is required.

Open M/SQL Program Structure

Open M/SQL programs are organized in units called **routines**. A routine is an individual block of code that is compiled, and then run. In Open M/SQL, all routine linking is dynamic at run time. The programmer never needs to link routines explicitly.

Open M/SQL routines can exist at three levels:

- Macro source level
- Intermediate code level
- Object code level

The three types of routines and their development and management in Open M/SQL are discussed below.

Macro Source Routines

Typically, you create routines at the macro source level. Macro source routines can include preprocessor syntax that permits macros, optional line inclusion, include files, and embedded SQL. Macro source routines are compiled into intermediate and object code routines.

All routines have macro source code, no matter what host M environment you are using.

Intermediate Code Routines

The middle level of routines is called intermediate code, which is the standard 3GL M source code available in all M implementations. The macro preprocessor phase of the Open M/SQL Compiler produces intermediate code from macro source code. At the intermediate code level, all preprocessor syntax is resolved, and the routine contains only pure M source code.

You may create, view, and edit routines at the intermediate code level using the Full Screen Editor. When you create routines at the intermediate code level, you must do so without the benefit of embedded SQL or any preprocessor syntax, such as macros.

Intermediate level source code is displayed on the terminal during debugging or error message display.

Object Code Routines

Object code is the lowest level of routine code. This is the code that is actually executed.

The Open M/SQL Compiler produces object code from intermediate code.

Routines can run in their object code form even if the macro source and intermediate code have been deleted. Each M implementation handles object code differently.

Open M/SQL Routine Utilities and Editors

Open M/SQL includes a number of routine utilities; some are designed to aid program development, while others assist system operations and maintenance:

- Routine development utilities provide editing, error analysis, date and time conversion, compilation, and maintenance capabilities.
- Developer utilities provide editing, object integrity checking, string search, and compilation capabilities.
- Global utilities provide analysis and control of the global database.
- System operation and maintenance utilities check hardware and database integrity, protect globals, and manage and control the database and networks.

Most Open M/SQL utilities are response driven. In other words, they use menus and prompt you for input. In addition, the utilities often include on-line documentation describing their uses and functions.

Open M/SQL provides the Full Screen Editor for use in creating and editing of macro source routines, include files, and intermediate code routines. A line-by-line editor for the editing and debugging of intermediate code routines is also available.

Routine Utilities for Non-ISM Implementations of M

Open M/SQL provides a set of utilities for editing Open M/SQL macro source routines. These routine utilities are available to all Open M/SQL users regardless of the host M environment.

In addition, for users of Open M/SQL running on non-ISM systems, the host M implementation will usually provide its own utilities for intermediate and object code management, global management, and system management.

Distributed Data Processing

Open M/SQL's distributed data processing feature enables you to link computer systems. From any system node, a user with appropriate access privileges can read and modify the global database anywhere in the system network. To reference a global on another system, you can:

- Issue the global reference with an extended syntax that specifies the directory and a system name for the other computer, or
- Use standard global syntax to reference an **implicit global** (a global that resides on another computer or in another directory on the same computer).

CHAPTER

2

The Open M/SQL Relational Database

Open M/SQL is based on an advanced relational model, which defines the database as a group of tables. InterSystems has extended the relational model to allow the definition of designative and characteristic relationships between tables. In Open M/SQL, any related group of tables that resides within an M data partition is considered a database. The M data partition is usually referred to as a directory or UCI, depending on your M system. An Open M/SQL database corresponds to the *schema* in the relational model.

Open M/SQL considers any related group of tables that resides within an M data partition to be a database. The M data partition is usually referred to as a directory or UCI, depending on your M system.

This chapter discusses relational concepts and their implementation in Open M/SQL.

Specifically, it covers the following topics:

- Open M/SQL Implements Relations as Tables
- Open M/SQL Implements Schemas as M Directories
- The Open M/SQL Data Dictionary
- Accessing Data in an Open M/SQL Relational Database
- InterSystems' Extensions to the Relational Model

Open M/SQL Implements Relations as Tables

A relational database is characterized by:

- A database structure that consists of a collection of **tables** (also called "relations" or "files").
- The presence of a query language that permits the manipulation of these tables in a **mathematically complete** manner.

The table, which is the basic unit of data storage in the relational model, is a collection of **rows** (also called "tuples" or "records") and **columns** (also called "fields" or "attributes"). Each column contains a particular type of data, such as integer, date, or text. (Columns may also contain null values.) Each row in a table corresponds to a real world entity and contains exactly one value for each column in the table.

Below is a sample relational table:

Table 2-1: A Sample Relational Table

EmpName	EmpNum	HireDate	Status	Salary
Grainger,Lisa	445-67-7891	06/06/90	Part-Time	28,000.00
Corson,Bob	210-92-8518	12/08/86	Active	15,000.00
Doe,Suzanne	333-44-7800	01/05/87	Active	38,000.00
Fast,Felix	334-45-5678	06/23/89	Active	44,000.00
Finley,Jack	356-62-1221	09/09/85	Part-Time	77,500.00
Gable,Bill	567-89-0123	08/02/92	Active	44,000.00

Open M/SQL Tables Follow The First Normal Form

Tables follow what is called the **First Normal Form**; that is, each column of a single row contains exactly one value, which may be null if the field is empty.

In some non-relational database systems, a column for a single row might contain several values. For example, the "Child" field in a "People" table might contain the names of three children; a single child field might also have three grandchildren.

However, in a relational system, each parent/child/grandchild relationship is represented by a separate row. For instance, each child of a given parent will occupy a unique row in the table despite the commonality of their parent fields. Likewise, each grandchild of that parent will also occupy its own unique row. This means that if a parent has three children who in turn have three children apiece, the relational model uses nine separate rows to describe the entire genealogy.

Alternatively, separate tables called characteristic tables (see below) can be used to designate relationships between parents, children, and grandchildren. The database designer must choose how s/he wants Open M/SQL to handle parent/child relationships.

Open M/SQL Supports Two Types of Tables

Open M/SQL supports two types of tables:

- Base tables
- Virtual tables

A base table is an autonomous, named table. Unlike virtual tables, base tables exist physically in the sense that they are mapped directly to physical storage structures. We say that base tables are “named” because the table is explicitly given a name via an appropriate definition statement, unlike, for example, the result of a query, which is not explicitly named and exists only ephemerally.

A virtual table is a named table derived from one or more base tables. Virtual tables are not directly represented in physical storage. Rather, they are abstract collections of base tables.

Examples of virtual tables include the output from a SELECT query and **views**. Views are windows through which data from multiple base tables can be “viewed”. Open M/SQL allows views to serve as data sources for M/PACT reports, SQL queries, and other views.

No two rows of a base table are identical. In virtual tables, two or more rows may be identical.

RowID/Primary Key

The **primary key** or **RowID** is a field or combination of fields that serve as the unique identifier to each row in a base table. At any given time, no two rows of the base table may contain the same primary key value.For example, the primary key of an “Employees” table might be the unique employee number; as each employee is identified by a unique employee number, each employee row in the table is identified by the value of the employee number field.

Since two rows of a base table cannot be identical while two rows of a virtual table can, only base tables have primary keys. Virtual tables (results of queries, for example) do not have primary keys.

In Open M/SQL, the RowID field is the primary key. The RowID is a single field of the table that uniquely identifies the row. Optionally, the database designer can base the RowID field on one or more other fields.

Open M/SQL Implements Schemas as M Directories

In the relational model, a schema is a conceptual repository for a group of relations. No relation within the schema can have the same name as another.

In Open M/SQL, the unit equivalent to the schema is an M database. In Open M/SQL, an M database can be spread over one or more directories. Each directory contains one component—MUMPS.DAT file—of the entire M database. A MUMPS.DAT file is usually referred to by the name of the operating system directory in which it resides. All tables defined in the same M directory must have unique names.

The system manager's directory contains a database directory table. The database directory table contains the database name, directory, and directory set if networked.

Open M/SQL Database Structure for Non-ISM Implementations of M

When Open M/SQL is layered on top of a non-ISM implementation of M, the database concept is implemented differently.

For example, under DSM, Open M/SQL considers all tables within a unique UCI and volume set to be a common database.

Other M systems employ various other schemes for storing the list of databases. For more information, see the documentation provided with your M system.

The Open M/SQL Relational Data Dictionary

In Open M/SQL, tables are defined and "mapped" to M global storage structures through the Open M/SQL Relational Data Dictionary. Mapping defines the link from the logical representation of a table to the physical storage of that table. When you define a database, the Relational Data Dictionary can generate a default global storage structure, or if you prefer, you may specify a customized global storage structure. The Relational Data Dictionary also lets you map existing global structures to Open M/SQL.

The Data Dictionary contains definitions for all the base tables that comprise the database. It also contains map specifications for the underlying data storage structures. The Form Generator, the SQL query facilities, and M/PACT all rely on Data Dictionary-defined structures. The window-based Data Dictionary makes the process of defining data structures very simple. It allows the programmer to develop applications using automated map generation facilities, and enables the linking of existing Open M/SQL applications, thus providing access to stored data through SQL, forms, and reports.

The following aspects of a database are defined in the Data Dictionary:

- Its conceptual elements (base tables, fields)
- RowIDs that enable data retrieval
- Global data storage ("map" specifications of global structures)
- Indexes for data access efficiency
- Implicit joins - characteristic relationships and designative references
- Triggers that enforce referential integrity and control application processing
- Views that control users' access to tables
- Data validation parameters for data insert and update
- Code to convert field values between data entry formats (external format) and database storage formats (internal format)
- Lookups for row selection in forms
- Field access code for forms
- Field display information for forms and reports

Accessing Data in an Open M/SQL Relational Database

Once you have defined tables in the Open M/SQL Relational Data Dictionary, you can access data from those tables by any of the following means:

- SQL queries — you may embed queries in M macro source code, or you may define them via the Open M/SQL Query Definition template or the Interactive SQL Query Editor
- M global references (in conjunction with or independent of SQL)
- Data entry, inquiry, and update forms designed using the Form Generator
- Reports defined and formatted using M/PACT

This section provides a brief overview of relational operations in SQL, the ANSI-Standard Query Language. SQL enables the retrieval of data from the relational database for the generation of queries and reports.

Refer to an SQL text for a full understanding of this language. Chapter 9, *Open M/SQL Implementation of SQL*, describes InterSystems' extensions to standard SQL.

Using SQL to Query the Database

Open M/SQL lets you query the database using standard SQL SELECT statement queries. Queries access data in tables and views. The output from a query forms a virtual table and leaves the targeted table(s) unchanged.

You can reference multiple tables in a single query.

SQL enables you to use the following relational operations when querying a database:

- Combining tables (or, more accurately, viewing tables as if they had been combined into one larger table)
- Selecting particular columns from a single table or a combined table
- Specifying particular rows in a single table or a combined table

Relational algebra provides the conceptual foundation for these relational operations. Understanding how the relational algebra works will help you take full advantage of the capabilities of SQL.

Conceptually, the relational algebra operates on one or more tables to produce a new (virtual) table. For example, a join combines two tables into a third. Or, a select operation extracts selected rows from one table to produce another table. A virtual output table is the result of the relational operation.

The list of fundamental relational operators includes:

- **Cartesian Product** — creates a cross-product of multiple tables, i.e. views the tables as if they had been combined into one larger table
- **Project** — selects particular columns from a single table or a combined table
- **Restrict** — selects particular rows from a single table or a combined table
- **Join** — selects some fields from some rows of multiple tables based on some relationship between the fields of the different tables

Table.Name Syntax

When referencing fields in an SQL query you may optionally precede the field name with the name of the base table to which it belongs, using the following syntax:

```
Table.Field
```

For example, a SELECT clause might read as follows:

```
SELECT Employees.EmpName, Depts.DeptName
```

If you do not use prefixes when creating field names that reflect the name of the base table in which they reside, it is generally a good idea to use this syntax for clarity.

If a query references multiple tables and those tables contain fields with identical field names, you must use the Table.Name syntax.

Cartesian Product

The **Cartesian Product** of two tables is the cross-product of all possible combinations of rows from the two tables, such that each row of the first table is combined (concatenated) with each row of the second table.

In SQL, the Cartesian Product operation occurs when two or more tables are explicitly or implicitly (within a view) named in the FROM clause, all fields in those tables are named in the SELECT clause, and no WHERE clause is present to specify relationships among the tables.

Example In the example below, the tables “Employees” and “Departments” are combined to form a virtual table, which is their Cartesian Product, by issuing the following SQL statement:

```
SELECT *
FROM Employees,Departments
```

The asterisk in the SQL SELECT statement is used to denote “all fields in the specified tables”.

The “Employees” table is shown below:

Table 2-2: Employees Table

EmpName	EmpNum	DeptNum
Bravo, Vicki	445-67-7800	1000
Doe, Suzanne	253-44-7898	3000
Corson, Bob	210-92-6518	2000

The “Departments” table is shown below:

Table 2-3: Departments Table

DeptNum	DeptName	NumEmp	DeptMgr
1000	Sales	50	Bravo, Vicki
2000	Administration	101	Corson, Bob
3000	Development	200	Doe, Suzanne

The table below represents the Cartesian Product of the “Employees” table and the “Departments” table:

Table 2-4: Output Table After Cartesian Product Operation

E.EmpName	E.EmpNum	E.Dept-Num	D.Dept-Num	D. DeptName	D.Num-Emp	D.DeptMgr
Bravo, Vicki	445-67-7800	1000	1000	Sales	50	Bravo,Vicki
Bravo, Vicki	445-67-7800	1000	2000	Administration	101	Corson, Bob
Bravo, Vicki	445-67-7800	1000	3000	Development	200	Doe, Suzanne
Corson, Bob	210-92-6518	2000	1000	Sales	50	Bravo,Vicki
Corson, Bob	210-92-6518	2000	2000	Administration	101	Corson, Bob
Corson, Bob	210-92-6518	2000	3000	Development	200	Doe, Suzanne
Doe, Suzanne	253-44-7898	3000	1000	Sales	50	Bravo,Vicki
Doe, Suzanne	253-44-7898	3000	2000	Administration	101	Corson, Bob
Doe, Suzanne	253-44-7898	3000	3000	Development	200	Doe, Suzanne

Note that this output may not be ordered by the “Employees.EmpName” field. The ordering is subject to the specifications in the ORDER BY clause of the SQL query, and

Project

The project operation extracts a subset of fields from an existing table. The result is a new table (a virtual table) with the same number of rows but fewer fields.

You can perform a Project operation on a single table or on a combination of tables.

Example In the example below, the fields “EmpName” and “DeptName” are selected from the Cartesian Product of the tables “Employees” and “Departments”.

```
SELECT Employees.EmpName,Departments.DeptName
FROM Employees,Departments
```

The SELECT clause specifies the fields to be included in the projected table. The Cartesian Product is derived from the tables named in the FROM clause.

This Project operation yields the following table:

Table 2-5: Output Table After Cartesian Product Operation

EmpName	DeptName
Bravo, Vicki	Sales
Bravo, Vicki	Administration
Bravo, Vicki	Development
Corson, Bob	Sales
Corson, Bob	Administration
Corson, Bob	Development
Doe, Suzanne	Sales
Doe, Suzanne	Administration
Doe, Suzanne	Development

Whether you perform a query on one or many tables, the conceptual result is always a new table that is derived from existing tables in the database.

Restrict

The Restrict operation selects a designated set of rows from one or more tables. In the SQL query language, restriction is expressed through the WHERE clause, which uses the comparison operations, such as >, <, and =, for example:

```
WHERE City="Boston"
```

or

```
WHERE Age>20
```

You can perform a Restrict operation on a single table or on a combination of tables.

Example In the example below, all fields are selected from the Cartesian Product of the tables "Employees" and "Departments", but the output table is restricted to only those rows for which the "NumEmp" field has a value greater than 100:

```
SELECT *
FROM Employees,Departments
WHERE Employees.NumEmp > 100
```

The SELECT clause uses the asterisk (*) to select all fields for inclusion in the table. The Cartesian Product is derived from the tables named in the FROM clause. The WHERE clause designates the condition for the restriction operation.

This Restrict operation yields the following table:

Table 2-6: Output Table After Restrict Operation

E.EmpName	E.EmpNum	E.Dept-Num	D.Dept-Num	D.DeptName	D.Num-Emp	D.DeptMgr
Bravo, Vicki	445-67-7800	1000	1000	Administration	101	Corson, Bob
Bravo, Vicki	445-67-7800	1000	3000	Development	200	Doe, Suzanne
Corson, Bob	210-92-6518	2000	1000	Administration	101	Corson, Bob
Corson, Bob	210-92-6518	2000	3000	Development	200	Doe, Suzanne
Doe, Suzanne	253-44-7898	3000	1000	Administration	101	Corson, Bob
Doe, Suzanne	253-44-7898	3000	3000	Development	200	Doe, Suzanne

Note that restriction occurs through the WHERE statement, not through the SELECT statement. The SELECT statement is used for projection, as discussed above.

Joins

Joins provide the means of linking data in one table with data in another table and are frequently used in defining reports and queries.

A join is an operation that combines two tables to produce a third, subject to a restrictive condition. Every row of the new table must satisfy the restrictive condition.

Usually, when the two tables (A and B) are combined to form a third table (C), some condition is specified in the WHERE clause. This condition determines how a row from B is chosen to combine with a row from A. Often, this condition is equality, such that the value of a particular field from table A equals the value from a particular field from table B. Combining tables in this way is called an equijoin. Equijoins are often referred to as inner joins.

Although joins are often thought of as fundamental operators, they represent a combination of Cartesian Product plus Restriction.

Example For an example of an inner join, consider the following two tables, a “Suppliers” table and a “Parts” table:

Table 2-7: Suppliers Table

SNum	SName	SCity
S1	Smith	Paris
S2	Jones	London
S3	Blake	Boston
S4	Whitney	Boston
S5	Roberts	Paris

Table 2-8: Parts Table

PNum	PName	PCity
P1	Nut	Paris
P2	Screw	Houston
P3	Cog	New York
P4	Wheel	Boston
P5	Switch	Boston

Suppose you wish to query the database for the names of every supplier, part, and city, where the supplier and part are located in the same city. To do this, you would use the following SQL query:

```
SELECT SName,PName,SCity
FROM Suppliers,Parts
WHERE Suppliers.SCity=Parts.PCity
```

The SELECT clause specifies the fields to be included in the projected table. The Cartesian Product is derived from the tables named in the FROM clause of the above SQL statement, which combines the tables “Suppliers” and “Parts”. The WHERE clause in the above statement (Suppliers.SCity=Parts.PCity) specifies the inner join condition.

The Cartesian Product of the “Suppliers” table and “Parts” table is shown below. The rows that satisfy the inner join condition are shaded.

Table 2-9: Output Table After Cartesian Product Operation

SNum	SName	SCity	PNum	PName	PCity
S1	Smith	Paris	P1	Nut	Paris
S1	Smith	Paris	P2	Screw	Houston
S1	Smith	Paris	P3	Cog	New York
S1	Smith	Paris	P4	Wheel	Boston
S1	Smith	Paris	P5	Switch	Boston
S2	Jones	London	P1	Nut	Paris
S2	Jones	London	P2	Screw	Houston
S2	Jones	London	P3	Cog	New York
S2	Jones	London	P4	Wheel	Boston
S2	Jones	London	P5	Switch	Boston
S3	Blake	Boston	P1	Nut	Paris
S3	Blake	Boston	P2	Screw	Houston
S3	Blake	Boston	P3	Cog	New York
S3	Blake	Boston	P4	Wheel	Boston
S3	Blake	Boston	P5	Switch	Boston
S4	Whitney	Boston	P1	Nut	Paris
S4	Whitney	Boston	P2	Screw	Houston
S4	Whitney	Boston	P3	Cog	New York
S4	Whitney	Boston	P4	Wheel	Boston

Table 2-9: Output Table After Cartesian Product Operation (Continued)

SNum	SName	SCity	PNum	PName	PCity
S4	Whitney	Boston	P5	Switch	Boston
S5	Roberts	Paris	P1	Nut	Paris
S5	Roberts	Paris	P2	Screw	Houston
S5	Roberts	Paris	P3	Cog	New York
S5	Roberts	Paris	P4	Wheel	Boston
S5	Roberts	Paris	P5	Switch	Boston

This join operation yields the following output table:

Table 2-10: Output After Join Operation

SName	SCity	PName
Smith	Paris	Nut
Blake	Boston	Wheel
Blake	Boston	Switch
Whitney	Boston	Wheel
Whitney	Boston	Switch
Roberts	Paris	Nut

When the join is based on an exact match between fields from the two tables, it is a simple join. Rows in the both tables where no match is found do not appear in the output table. There are no rows in the output table for the cities, “London”, “New York”, or “Houston”.

InterSystems' Extensions to the Relational Model

InterSystems has extended SQL and the relational model to include two additional types of joins: **one-way outer joins** and **implicit joins**.

One-Way Outer Joins

With standard "inner" joins, when rows of one table are linked with rows of a second table, a row in the first table that finds no corresponding row in the second table is excluded from the output table.

With one-way outer joins, all rows from the first table are included in the output table even if there is no match in the second table. The first table pulls relevant information out of the second table but never sacrifices its own rows for lack of a match in the second table.

When specifying a one-way outer join, the order in which you name the tables in the FROM clause is very important. The first table you specify is the source table for the join.

You specify an outer join by using the symbol =* in place of = in the WHERE clause of the SQL query.

Example In the example below, the "Suppliers" table is specified as the source table for a one-way outer join operation (=*) with the "Parts" table, where the "SCity" field matches the "PCity" field.

```
SELECT SName,PName,SCity
FROM Suppliers,Parts
WHERE Suppliers.SCity=*Parts.PCity
```

The SELECT clause specifies the fields to be included in the projected table. The Cartesian Product is derived from the tables named in the FROM clause of the above SQL statement, which combines the tables "Suppliers" and "Parts". The WHERE clause in the above statement (Suppliers.SCity=*Parts.PCity) specifies the one-way outer join condition.

This query returns all rows from the "Suppliers" source table as well as any rows from the "Parts" table where the "SCity" field matches the "PCity" field.

This join operation yields the following output table:

Table 2-11: Output After Join Operation

SName	SCity	PName
Smith	Paris	Nut
Jones	London	
Blake	Boston	Wheel
Blake	Boston	Switch
Whitney	Boston	Wheel
Whitney	Boston	Switch
Roberts	Paris	Nut

Implicit Joins

One of the most powerful features of the relational model is its ability to handle unanticipated ad hoc queries in a graceful and straightforward manner. However, for many implementations this capability is costly: the power to join tables in complex ways often means sacrificing the ease of executing joins for the more common connections between tables.

InterSystems has solved this problem by implementing the **implicit join**. Implicit joins are pre-defined joins between tables which you specify in the Data Dictionary. They allow you to define queries without specifying the WHERE condition that is used to join tables.

Open M/SQL supports two types of implicit joins, **designative references** and **characteristic relationships**.

Designative references and characteristic relationships are useful for:

- Pre-defining commonly used joins
- Improving data access efficiency
- Formally specifying integrity constraints

Note You may only define designative and characteristic relationships among tables that reside within a single database.

Designative References

A designative reference is a many-to-one link between tables in which one field of the designating table contains the Row IDs of all rows in the designated table. A designative reference is said to be a non-dependent link because rows in the referenced table exist independently of rows in the designating table. In relational database terminology, the designating table has a “foreign key” on the referenced table. In M terminology, the designating table has a “pointer” to the referenced table. In Open M/SQL, a field that designates another table is called a designative reference field.

In the example below, the “Customer” field of the “Invoice” table serves as the Designative Reference field to rows in the “Customer” designated table:

Table 2-12: Invoice Table

InvNum	InvTotal	Customer
1234	100.00	C1
5555	20,000.00	C3
3333	5,000.00	C4

Table 2-13: Customer Table

Number	Name	Address
C1	Acme Hardware	10 Main Stree Boston, MA
C2	Waterfront Motors	210 Willow Street Brighton, MA
C3	Global Furniture	1010 5th Street New York, NY
C4	Hill Pharnaceuticals	958Jordan Ave. Pittsburgh, PA

The Designative Reference field, when it is not empty, contains a value that identifies one and only one row of the referenced table. Every entry in the “Customer” field of the “Invoice” table that is not empty must have exactly one corresponding entry in the “Customer” table. However, not all of the “Number” values in the “Customer” table need appear in the “Customer” field of the “Invoice” table. In this way, a designative reference satisfies the relational definition of a one-way outer join.

Furthermore, when the "Invoice" table is linked by designative reference to the "Customer" table, the following is true:

- There may be invoices with no "Customer" value, but
- If a "Customer" value appears, there must also be a "Number" value in the "Customer" table with the same value, and
- There may be customers with no invoices

Designative references contribute to data storage efficiency by helping to eliminate unnecessary redundancy. Since you can access the information through a designative reference to the "Customer" table, it is not necessary to store the customer names and addresses in the "Invoice" table. Accordingly, updates need be made in only one table, rather than in two or more tables.

A table may have several fields that designate the same or different tables. Similarly, a table may be designated by any number of tables. For example, the "Invoice" table may contain another designative reference field to the "Accounts" table. And the "Accounts" table might have a designative reference to the "Customer" table.

Characteristic Relationships

A characteristic relationship is a link between tables in which rows in one table (the "child table") are existence-dependent (cannot exist without) on rows in another table (the "parent table"), such that parent rows have a one-to-many relationship with child rows. A child table always designates its parent table. For this reason, a characteristic relationship can be thought of as a kind of designative reference. However, a characteristic relationship is more restrictive than a designative reference since the join condition specifies that all rows of the child table must designate the same parent table row. In this way, a characteristic relationship satisfies the relational definition of an inner join.

Extending our previous example, the "Line Items" table, shown below, is a child table of the "Invoice" table (its parent). The existence of the "Line Items" table is entirely dependent on its parent "Invoice" table. If an invoice is deleted from the parent table, its line items become 'orphaned' and must also be deleted, or transferred to a "foster parent". Within the Data Dictionary definition of a table, you may set a trigger that will function to automatically delete all child rows when its parent is deleted.

Each row of the “Invoice” table can have multiple line items, illustrating the one-to-many relationship:

Table 2-14: Invoice Table

InvNum	InvTotal	Customer
2222	10,000.00	C3
5555	20,000.00	C2
1234	100.00	C1

Table 2-15: Line Items Table

InvNum	Item	UnitCost	Quantity	Amount
2222	Chair	200.00	10	2,000.00
2222	Desk	300.00	10	3,000.00
2222	Rug	250.00	4	1,000.00
2222	Bookshelf	200.00	30	6,000.00

An “Invoice” row might also have no line items.

Though characteristic relationships may be perceived as hierarchical, they comply with the tenets of the relational model; just as other tables, child tables can be addressed through SQL without explicit reference to the parent table.

A parent table may have several child tables. For example, a “Patient” table may have “Visits”, “Medical Problems”, and “Lab Tests” as child tables. However, a child table may have only one parent. (The “Visits” table, for example, may be a child only of the “Patient” table.) A child table can never be 'orphaned', i.e. exist without a parent table.

Implicit Join Syntax

Implicit joins simplify the process of querying the database. By defining characteristic relationships and designative references in the Data Dictionary, you may take advantage of Open M/SQL's implicit join syntax to facilitate the definition of queries.

Arrow syntax, a dash followed by a greater-than symbol (->), is an InterSystems' SQL extension used to indicate an implicit join between tables. This syntax causes an additional outer join condition to be added implicitly to the WHERE clause and the joined table to be added implicitly to the FROM clause of an SQL query.

In Open M/SQL, you may use arrow syntax in the following three cases:

1. To signify Designative References between tables
2. To signify Child-to-Parent References between tables
3. To signify Parent-to-Child References between tables

Implicit Join Syntax in a Designative Reference

If the field A.b designates table B, and x is a field in table B, the reference:

```
A.b->x
```

points to the value of x in the row of table B corresponding to A.b. It is interpreted as a reference to B.x with B added implicitly to the FROM clause and an additional outer join condition added implicitly to the WHERE clause.

For example, the following query retrieves the patient's name and patient's doctor's name for every patient who lives in Boston:

```
SELECT Patient.Pname, Patient.Doctor->Dname
FROM Patient
WHERE Patient.City = "Boston"
```

Assuming that "Patient.Doctor" is a designative reference to the "Doctor" table, the above query is equivalent to:

```
SELECT Patient.Pname,Doctor.Dname
FROM Patient,Doctor
WHERE Patient.City = "Boston"
      AND Patient.Doctor = *Doctor.Doctor
```

Implicit Join Syntax in a Child-to-Parent Reference

If P is the parent of C and x is a field in P, the implicit join syntax:

```
C.P->x
```

for a given row points to the value of x in that row's parent row. It is interpreted as a reference to P.x with P added implicitly to the FROM clause and an additional outer join condition added implicitly to the WHERE clause.

For example, given a parent table “Customer” with a child table “Invoice”, the following query:

```
SELECT Invoice.Customer->Name
FROM Invoice
WHERE Invoice.Number = 51140
```

is equivalent to:

```
SELECT Customer.Name
FROM Invoice,Customer
WHERE Invoice.Number = 51140
   AND Invoice.Customer = Customer.Customer
```

Implicit Join Syntax in a Parent-to-Child Reference

If P is the parent of C and x is a field in C, the implicit join reference:

```
P.C->x
```

for a given P row points to the value of x in a child row of that row. It is interpreted as a reference to C.x with C added implicitly to the FROM clause and an additional outer join condition added to the WHERE clause.

For example, given parent “Customer” with child “Invoice”, the following query for all invoices for all customers named Smith:

```
SELECT Customer.Invoice->Number
FROM Customer
WHERE Customer.Name = “Smith”
```

is equivalent to:

```
SELECT Invoice.Number
FROM Customer,Invoice
WHERE Customer.Name = “Smith”
   AND Customer.Customer = Invoice.Customer
```

Integrity Constraints

Implicit join definitions include built-in integrity constraints. For instance, the existence-dependent relationship between the "Line Items" table and the "Invoice" table may be regarded as an integrity constraint: no line item can exist without a corresponding invoice.

The Open M/SQL relational Data Dictionary can be used to define other integrity constraints, such as:

- Field validation code to enforce integrity constraints at the field processing level, such as affect field values, required fields, and field formats.
- Triggers to enforce table integrity constraints, such as complex interactions between fields, or the prohibition of DELETEs from the table.

Multi-Line Fields

Open M/SQL extends the relational database model by permitting the creation of multi-line fields. Such fields are useful for storing information about a single entity where that information spans several lines. A typical multi-line field might be used for an address or a block of comment text.

Open M/SQL treats the data in multi-line fields as a single entity, in accordance with First Normal Form principle of the relational model.

InterSystems' SQL supports the use of multi-line fields in input operations (INSERT and UPDATE statements) and output operations using INTO lists by creating an array and matching each line of the multi-line field to a node in the array. InterSystems' SQL also supports the naming of multi-line fields in the SELECT statement of SQL SELECT queries. It does not, however, allow the use of multi-line fields to perform comparisons or row ordering in an SQL SELECT query.

You can access pieces of multi-line field data using the M language, but this use of multi-line fields is not recommended, because it does not adhere to first normal form.

PART

II

Program Development

Chapter 3
Open M/SQL Program Development

Chapter 4
Full Screen Editor

Chapter 5
Developing Macro Source Routines

Chapter 6
Routine Handling and Maintenance

Chapter 7
Open M/SQL Routine Management Utilities

CHAPTER

3

Open M/SQL Program Development

This chapter presents a brief overview of the facilities available for program development in Open M/SQL.

Specifically, it covers the following topics:

- Programming Methods
- Full Screen Editor
- Intermediate Code Routines
- Routine Management Utilities
- Developer Utilities
- Programmer Interface to Applications

Programming Methods

Open M/SQL allows the programmer to develop hand-coded applications at two levels: the macro source level and the intermediate code level.

Typically, you create routines at the macro source level. At the macro source level, you may define macros, refer to existing macros, and write pure M code or embed SQL statements using a combination of ANSI-Standard M syntax, special macro preprocessor commands, and ANSI-Standard SQL. Macro source code also makes use of include files. Like macro source code, include files may contain M syntax, SQL syntax, and preprocessor syntax.

The macro preprocessor phase of the Open M/SQL Compiler converts macro source code into M code with an internal form of embedded SQL. This converted code is called intermediate code. You may view and edit routines at the intermediate code level. You may also create routines directly at the intermediate code level, although without the benefit of embedded SQL or preprocessor syntax, such as macros. One strategy you may use is to create pure M routines at the intermediate code level, copy those routines to the macro source level using the %urcopy utility, then edit the routines in the Full Screen Editor to include preprocessor syntax and embedded SQL.

The Full Screen Editor

You may use the Full Screen Editor to edit any of the following types of routines:

- Macro source routines
- Include files
- Intermediate code routines

You must use the Full Screen Editor to create and edit macro source routines and include files.

You may create and edit intermediate code routines either using the Full Screen Editor or directly from the M prompt using the Routine Line Editor.

Intermediate Code Routines

Open M/SQL provides three ways to create intermediate code routines:

1. Intermediate code routines are the products of compiled macro source code routines — these routines consist of M code with the possible inclusion of embedded SQL statements.
2. You may create intermediate code routines in the Full Screen Editor — these routines consist of pure M source code.
3. You may create intermediate code routines at the M programmer mode prompt — these routines consist of pure M source code.

Routine Line Editor

The Routine Line Editor lets you edit and debug intermediate code routine lines and insert new routine lines directly from the M programmer prompt. The Routine Line Editor operates only on intermediate code—it does not operate on macro source code or include files. For more information on using the Routine Line Editor to develop and edit intermediate code routine lines, see the *Open M/SQL M Programming Guide*.

Routine Management Utilities

InterSystems provides a set of pre-defined utilities for examining and manipulating routines and include files. These routine utilities are useful for developing and maintaining Open M/SQL applications. See Chapter 7, *Open M/SQL Routine Management Utilities,* for a complete description of the Open M/SQL routine management utilities.

Developer Utilities

InterSystems provides a set of utilities useful for testing and developing programs and manipulating program objects. These utilities perform functions that include checking the integrity of program objects, searching for strings in objects, and compiling sets (called configurations) of objects. For a complete description of the Open M/SQL developer utilities see Chapter 12, *Open M/SQL Developer Utilities.*

Programmer Interface to Applications

Open M/SQL combines the precision of hand-coded programming with the speed and ease of application generation to provide a totally integrated application development environment.

You can reference Data Dictionary-defined global structures using any combination of SQL and M code. Open M/SQL provides entry points that enable you to call menus, forms, and reports from anywhere in your application. You can insert M and SQL code directly into base table definitions, form definitions, menu definitions, and report definitions, which allows you to develop highly customized applications while still preserving Open M/SQL's automatic-generation capabilities.

In the Data Dictionary, you can define processing triggers to enforce integrity constraints or to automatically invoke related processing actions every time a certain event occurs. You can enter code to perform data validation checking and conversion of user-entered and edited data. You can define computed fields that will automatically calculate data values based on user input. And you can manually insert your own M code to customize lookup and filing routines.

In the Form Generator, you can manually insert M and SQL code at the form level, window level or field level to customize your application to your exact specifications. You can use processing triggers to program window branching that responds in different ways to different situations.

These are but a few of the many ways the Open M/SQL environment combines custom programming flexibility with the ease of automatic generation.

CHAPTER

4

Full Screen Editor

This chapter describes the Open M/SQL Full Screen Editor. The Full Screen Editor is used for creating, editing, and viewing macro source routines, intermediate code routines, and include files.

Specifically, this chapter covers the following topics:

- Overview of the Full Screen Editor
- Invoking the Full Screen Editor
- Full Screen Editor Screen Display
- Navigating the Full Screen Editor Menu System
- Editing Operations
- Getting Help
- Exiting the Full Screen Editor

Overview of the Full Screen Editor

The Full Screen Editor is an Open M/SQL utility that allows you to view an entire block of source code and edit sections of it. By contrast, the alternative editing utility, the Routine Line Editor, allows only line-by-line editing (see the *Open M/SQL M Programming Guide* for an in-depth description of the Routine Line Editor).

Full Screen Editor Features

The Full Screen Editor provides all of the following capabilities:

- Cut and paste capabilities
- Search and replace functions
- Ability to work in more than one buffer or window at a time
- Automatic syntax checking
- A lockout mechanism to prevent overwrites

When you edit a routine in the Full Screen Editor, you are editing a temporary copy of the routine. Pressing the <PREVIOUS> key invokes a save menu that displays options for saving, compiling, and renaming the routine as well as an option for exiting the Full Screen Editor.

Routine Types for Editing

Using the Full Screen Editor, you may edit any of the following types of routines:

- Macro source routines
- Include files
- Intermediate code routines (ANSI Standard M)
- The routine in your current M partition

Invoking the Full Screen Editor

You can invoke the Full Screen Editor from either the M programmer prompt or from within the Open M/SQL development environment.

Procedure **To invoke the Full Screen Editor from the M programmer prompt:**

1. **Issue any of the following commands at the M programmer prompt:**

```
> do ^%rde
```

or:

```
> do ^%
```

or:

```
> x ^%
Edit: .E
```

To exit from the "Edit:" prompt befor invoking the Full Screen Editor, type a period (.), and press <RETURN>.

Note If you are running Open M/SQL on a non-ISM host M system, you must use the first command (^%rde).

Once invoked, the Full Screen Editor prompts you to load a routine, as follows:

```
Load Routine:
```

At this prompt you may either load an existing routine or create a new routine. See below for information on both options.

Procedure **To invoke the Full Screen Editor from the Open M/SQL development environment:**

1. **At the M programmer prompt, type the following command to enter Open M/SQL:**

   ```
   > do ^%msql
   ```

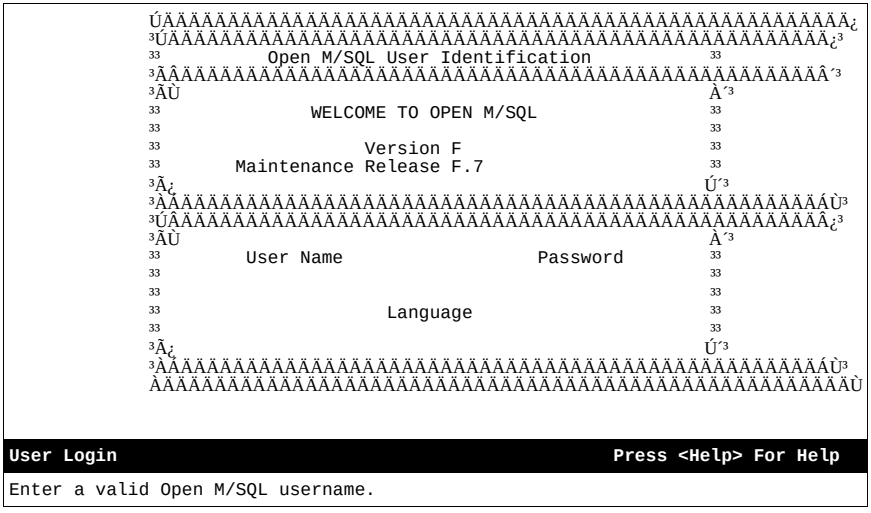
   You see the Terminal Type prompt, as shown below:

   ```
   Terminal Type: VT220 =>
   ```

2. **At the Terminal Type prompt, enter the name of the terminal type you are currently using.**

   You may press <RETURN> to accept the system-wide default terminal type.

   You see the Open M/SQL User Identification window, as shown below:



3. **At the UserName field on the Open M/SQL User Identification window, enter your Open M/SQL UserName, and press <RETURN>.**

4. **At the Password field on the Open M/SQL User Identification window, enter the Password for your Open M/SQL UserName, and press <RETURN>.**

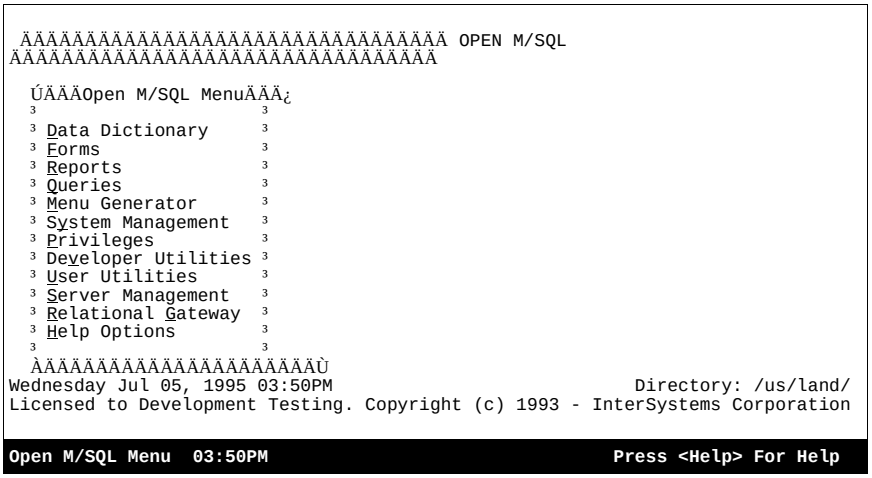
5. **At the Language field on the Open M/SQL User Identification window, enter the language in which you want to run Open M/SQL.**

   To accept the system-wide default run-time language, press <RETURN>.

   To choose a different run-time language, press <CTRL-L> to delete the system-wide default language.

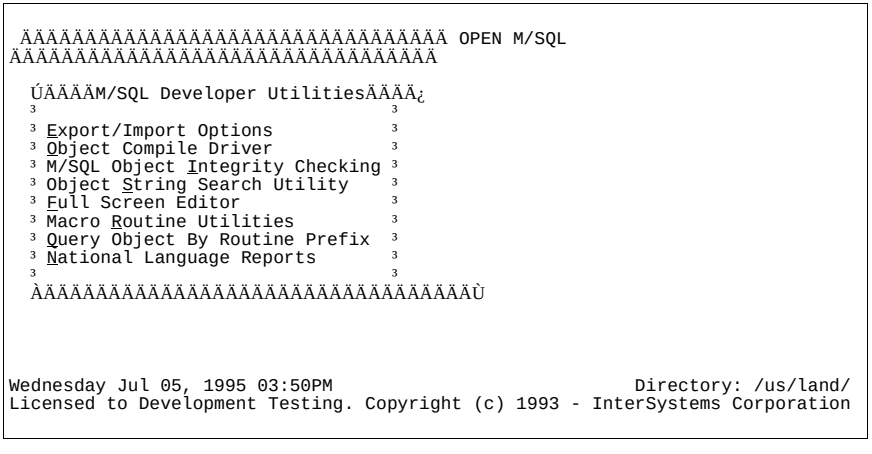
   You may press the <LIST CHOICES> key to see a lookup box that lists the run-time languages supported by Open M/SQL.

   You see the Open M/SQL Main Menu, as shown below:



6. **From the Open M/SQL Main menu, select the Developer Utilities option.**

   You see the M/SQL Developer Utilities menu, as shown below:



7. **From the M/SQL Developer Utilities menu, select the Full Screen Editor option to invoke the Full Screen Editor.**

   Note: You may type *f* to select this option—it is a mnemonic accelerator.

   Once invoked, the Full Screen Editor prompts you to load a routine, as follows:

   ```
   Load Routine:
   ```

   At this prompt you may either load an existing routine or create a new routine. See below for information on both options.

Loading Existing Routines

To load an existing routine, enter the name and extension of the routine or include file you wish to edit at the "Load Routine" prompt.

When entering the name of a routine, if you do not specify a suffix, Open M/SQL assumes the .MAC suffix.

For example, to load the routine ABC.MAC, you may type the following:

```
Load Routine: ABC
```

If you specify the name of a routine that already exists, Open M/SQL loads the routine and invokes the Full Screen Editor.

Loading Routines Automatically

You can set up Open M/SQL to automatically load a specified routine whenever you invoke the Full Screen Editor. You do this by setting the %first variable to the routine that you want to be auto-loaded.

Procedure **To load a specified routine automatically:**

1. **From the M programmer prompt, set the %first variable to the routine that you want to be auto-loaded.**

   Use the following syntax to do this:

   ```
   > SET %first="routine.ext"
   ```

   where *routine.ext* is the name and extension of the routine you want to autoload.

2. **Invoke the Full Screen Editor.**

The Full Screen Editor automatically loads the routine stored in the %first variable.

Creating New Routines

To create and load a new routine, use the procedure below:

Procedure **To create a new routine using the Full Screen Editor:**

1. **At the "Load Routine" prompt, enter the name of a new routine, as follows:**

   ```
   Load Routine: NEWROU
   ```

   The Full Screen Editor automatically appends the default extension ".MAC" to the routine name you entered and then issues the following prompt:

   ```
   Load Routine: NEWROU
    the source 'NEWROU.MAC' does not exist
   Continue with new buffer? Y=>
   ```

2. **Press <RETURN> to accept the Yes default and load the new routine into the Full Screen Editor, or type *No* and press <RETURN> to return to the "Load Routine" prompt.**

Preventing Overwrites

The Full Screen Editor provides a lockout mechanism to ensure that two programmers cannot modify the same routine simultaneously. This mechanism issues a lock on a routine as soon as the first user retrieves it.

The locking mechanism prevents the following scenarios from occurring:

- Prevents two users from editing the same .MAC file simultaneously
- Prevents two users from editing the same .INT file simultaneously
- Prevents two users from editing the same .INC file simultaneously
- Prevents one user from editing the .MAC version of a routine while another user is simultaneously editing the .INT version of the same routine, or vice versa.

When the Full Screen Editor detects a lock on a routine, it allows the second user to browse the routine in "read mode", but the user cannot edit or file the routine. When this happens, the terminal beeps and the Full Screen Editor displays the following message at the bottom of the screen:

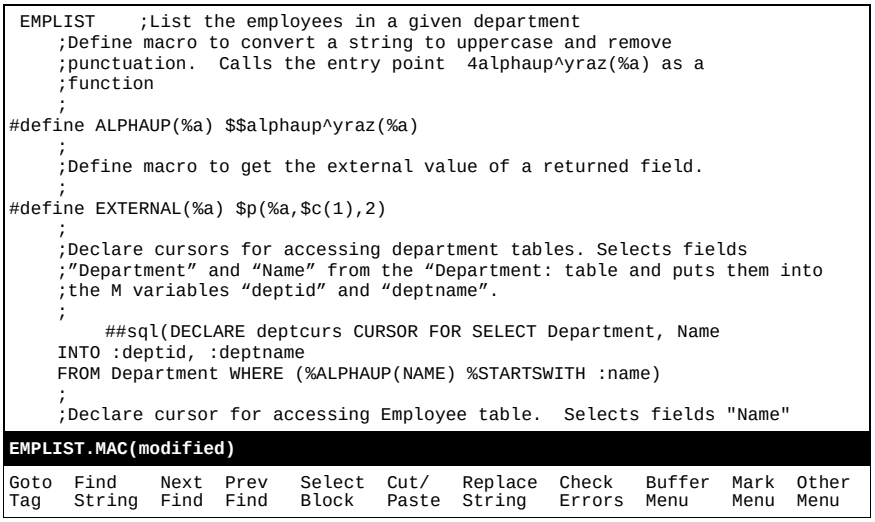
```
ROUTINE IS LOCKED: Now in Read Mode
```

As you browse the locked routine, the Status Line continues to display the tag:

```
(read)
```

Full Screen Editor Screen Display

Below is a typicat Full Screen Editor screen display:



The screen display is divided into the following three parts:

- Edit Field — the top (and major) portion of the screen
- Status Line — Directly below the Edit Field
- Horizontal Options Menu — Directly below the Status Line

Edit Field

The Full Screen Editor display consists of as many lines as will fit on your terminal screen. One of those lines is reserved for the Status Line and two more are reserved for the Horizontal Options Menu. The remaining lines belong to the Edit Field.

The typical terminal screen consists of 24 lines, therefore the Edit Field of a typical terminal screen displays up to 21 lines of text.

The Full Screen Editor is always in insert mode.

The Edit Field scrolls appropriately as you insert and delete text.

Status Line

The Status Line appears in reverse video between the Edit Field and the Horizontal Options Menu. It displays the following information:

- Name of the current buffer
- Name of the current routine
- Type of routine being edited (.MAC, .INT, .INC)

When changes have been made to the current routine, the Status Line displays the following tag:

```
(modified)
```

If the current routine is non-modifiable, the Status Line displays the tag:

```
(read)
```

The information displayed on the Status Line is standard and does not vary across systems. The system, however, may alter the appearance of the information. The standard appearance format is as follows:

```
[buffer]routine_name.type
```

Horizontal Options Menu

The Horizontal Options Menu displays across the bottom of the screen, directly beneath the Status Line. You can use the options on this menu to perform basic editing functions such as selecting and moving text.

Procedure **To access an option on the Horizontal Options Menu:**

1. **From anywhere in the Edit Field of the Full Screen Editor, press the <GO TO BOTTOM MENU> key.**

   The cursor moves to the Horizontal Options Menu.

2. **On the Horizontal Options Menu, you may select and invoke an option using any one of the following methods:**

   a. Use the arrow keys to position the cursor on the desired option, and press <RETURN> to invoke it.

   b. Type the first letter of an option (the cursor selects and automatically invokes the option).

   c. Press <CTRL-E> plus the first letter of the desired option.

   d. If you are selecting an option from a submenu, you may type <CTRL-E>*xy* where *x* is the first letter of the primary menu option and *y* is the first letter of the submenu option.

Navigating the Full Screen Editor Menu System

The following sections list and describe the options located on each of the horizontal options menus in the Full Screen Editor menu system

Primary Menu

The following table lists and describes the options on the Primary Menu:

Table 4-1: Options on Primary Menu

Option	Function
Goto Tag	Specify the tag and offset within your M routine that you wish to locate, using the syntax: ***TAG*** ***TAG+3*** ***TAG-3***
Find String	Specify a string to be located.
Next Find	Finds next occurrence of a string specified in previous Find, working towards the bottom of the buffer.
Prev Find	Finds previous occurrence of a string specified in a previous Find, working towards the top of the buffer.
Select Block	Turn on select mode. Any cursor movement while in select mode results in highlighting the text between the current cursor position and its position when select mode was activated. Selecting this option a second time eliminates the highlighting.
Cut/Paste	If select mode is on, cuts the contents of the select region from the text and inserts it into the paste buffer, then turns off select mode. If select mode is off, inserts the contents of the paste buffer into the current buffer.
Replace String	Scans the buffer for a specified string, and optionally replaces it with another.
Check Errors	Syntax checks the routine from the location of the cursor down, replacing the cursor at the beginning of the first line in which it detects a syntax error. This is an error-by-error syntax checking mechanism.
Buffer Menu	Invokes Buffers submenu (described below).
Mark Menu	Invokes Mark submenu (described below).
Other Menu	Invokes Other submenu (described below).

Buffers Menu

When you select the Buffers Menu option from the Primary Menu, you see the Buffers Menu. The table below lists and describes the options on the Buffers Menu:

Table 4-2: Options on Buffers Menu

Option	Function
Make Buffer	Make a new buffer with a specified routine name and load that routine if it exists.
Use Buffer	Switch from one buffer to another.
Directory of Buffers	View the names of all existing buffers.
Print Buffer	Print the contents of buffer to a device.
Load Routine	Load a routine into current buffer, deleting the existing contents of the buffer.
Insert Routine	Insert a routine into current buffer, without deleting the existing contents.
Windows Menu	Invokes Windows menu.

Windows Menu

When you select the Windows Menu option from the Buffers Menu, you see the Windows Menu. The table below lists and describes the options on the Windows Menu:

Table 4-3: Options on Windows Menu

Option	Function
Two Windows	Arranges the terminal screen to display two buffers at the same time, each in its own area (window).
One Window	Sets the terminal screen to display one buffer using the entire screen area.
Switch Windows	Moves the cursor from one buffer window to the other.
Grow Current Window	Increases the size of one of the two displayed windows, allowing it to occupy a greater portion of the screen.

Mark Menu

When you select the Mark Menu option from the Primary Menu, you see the Mark Menu. The table below lists and describes the options on the Mark Menu:

Table 4-4: Options on Mark Menu

Options	Function
Set Mark	Mark a location in the current buffer.
Clear Mark	Remove a mark from the current buffer.
Find Mark	Go to a specified mark in the current buffer.

Other Menu

When you select the Other Menu option from the Primary Menu, you see the Other Menu. The table below lists and describes the options on the Other Menu:

Table 4-5: Options on Other Menu

Option	Function
Redraw Screen	Redraws screen without saving or deleting contents of buffer.
Execute M Code	Prompts for lines of M code and executes them. Type “Q” to exit from this option.
M/SQL	Invokes Open M/SQL

Editing Operations

This section describes how to perform the following text editing operations:

- Moving the Cursor
- Inserting and Deleting Text
- Cutting and Pasting Text
- Editing Multiple Copies of a Routine
- Displaying Multiple Buffers
- Setting a Mark in Your Current Buffer
- Searching For Text Strings
- Replacing Text Strings

Moving the Cursor

The use of control keys simplifies cursor movement for touch typists. The following table describes the cursor positioning keys available for use in the Full Screen Editor:

Table 4-6: Full Screen Editor Cursor Movement Key Commands

Keystroke(s)	Function
<RIGHT ARROW> or <CTRL-K>	Cursor moves one character to the right. If at right physical margin, cursor moves to the first character of the next line.
<LEFT ARROW> or <CTRL-H>	Cursor moves one character to the left. If at left physical margin, cursor moves to the last character of the previous line.
<UP ARROW> or <CTRL-U>	Moves the cursor to the same column position on the next physical line up.
<DOWN ARROW> or <CTRL-J>	Moves the cursor to the same column position on the next physical line down.
<ENHANCE><RIGHT ARROW>	Moves the cursor to the end of the current M code line.
<ENHANCE><LEFT ARROW>	Moves the cursor to the beginning of the current M code line.
<ENHANCE><UP ARROW>	Moves the cursor to the first character of the edit field.
<ENHANCE><DOWN ARROW>	Moves the cursor to the first character of the last line of the edit field.
<CTRL-N>	Moves the cursor to the next tag.
<CTRL-P>	Moves the cursor to the previous tag.
<CTRL-V>	Moves the cursor to the last character of the routine being edited (bottom of routine).

Table 4-6: Full Screen Editor Cursor Movement Key Commands (Continued)

Keystroke(s)	Function
<CTRL-F>	Moves the cursor to the first character of the next word*.
<CTRL-B>	Moves the cursor to the last character of the previous word*.
<CTRL-E><1>	Repaints the screen when it is split into two windows for the syntax error report.

* A word is defined as any sequence of characters delimited by one or more spaces or commas.

Inserting Text

The Full Screen Editor is always in insert mode.

Procedure **To insert text:**

1. **Use the arrow keys to locate the cursor at the place where you wish to insert text.**
2. **Begin typing.**

   Note: There is an implicit hard return at the end of each M code line. Pressing <RETURN> breaks the line. Deleting a <RETURN> joins the line.

DeletingText

To delete text, use the appropriate key or key sequence from the following table:

Table 4-7: Text Delete Options

Keystroke	Function	Description
<CTRL-D>	Delete Current Character	Deletes the character on which the cursor is currently positioned.
<DELETE>	Delete Previous Character	Deletes the character to the left of the current cursor position.
<CTRL-W>	Delete Word	Deletes from the current cursor position to the end of the current word.
<CTRL-L>	Delete to End of Line	Deletes from current cursor position to the end of the current physical screen line.
<CTRL-X>	Undelete	Restores a previously deleted character, word, or line.

Cutting and Pasting Text

To cut and paste text, use the following procedure.

Procedure **To cut and paste text:**

1. **Place the cursor at the beginning of the block of text that you want to cut.**
2. **Press the <GO TO BOTTOM MENU KEY> to access the Horizontal Options Menu.**
3. **From the Horizontal Options Menu, choose the Select Block option.**

   The Select option makes the cursor function as a tool for highlighting blocks of text.
4. **Move the cursor to the end of the block of text that you want to select.**

   The cursor highlights all text in its path from the location where Select mode was activated.
5. **Press the <GO TO BOTTOM MENU KEY> to access the Horizontal Options Menu.**
6. **From the Horizontal Options Menu, select the Cut/Paste option to cut the text.**

   When you cut text from the document, the Full Screen Editor stores it in an area called the Paste Buffer. Each time you cut a block of text, the new text overwrites the previous contents of the Past Buffer. The Paste Buffer always contains the most recently cut block of text.
7. **Move the cursor to the location where you wish to insert the text.**
8. **Press the <GO TO BOTTOM MENU KEY> to access the Horizontal Options Menu.**
9. **From the Horizontal Options Menu, select the Cut/Paste option again to paste the text.**

Editing Multiple Copies of a Routine

The Full Screen Editor provides buffers and windows to let you edit multiple routines and multiple copies of the same routine simultaneously.

Buffers are conceptual spaces where you can temporarily store text to be edited. The first routine loaded into the Editor is automatically placed in a buffer called MAIN. You can then create and name additional buffers, and place new text into the new buffers. You can also create empty buffers for use during an editing session.

When placing routines in buffers, you can LOAD or INSERT. LOAD places the text into the buffer, overwriting the current contents of the buffer. INSERT places the designated text into the buffer without deleting the current contents.

Creating a New Buffer

Use the procedure below to create a new buffer.

Procedure **To create a new buffer:**

1. **Press the <GO TO BOTTOM MENU KEY> to access the Horizontal Options Menu.**
2. **From the Horizontal Options Menu, select the Buffer Menu option.**

   You access the Buffers Menu.
3. **From the Buffers Menu, select the Make Buffer option.**

   You see the following prompt in the bottom right-hand corner of the screen:

   ```
   Make Buffer:
   ```
4. **At the "Make Buffer" prompt, enter the name of a new buffer, and press <RETURN>.**

   The Full Screen Editor opens a new buffer and places you in insert mode.

Selecting an Existing Buffer

Use the procedure below to select an existing buffer.

Procedure **To select an existing buffer:**

1. **Press the <GO TO BOTTOM MENU KEY> to access the Horizontal Options Menu.**
2. **From the Horizontal Options Menu, select the Buffers Menu option.**

   You access the Buffers Menu.

3. **From the Buffers Menu, select the Use Buffer option.**

   You see the following prompt in the bottom right-hand corner of the screen:

   ```
   Use Buffer:
   ```

4. **At the "Use Buffer" prompt, enter the name of the buffer you want to edit, and press <RETURN>.**

   Note: To see a list of all current buffers, select the Directory option from the Buffer Menu.

   The Full Screen Editor opens the specified buffer and places you in insert mode.

Loading a Routine into the Current Buffer

Use the procedure below to load a routine into the current buffer.

Procedure **To load a routine into the current buffer:**

1. **Press the <GO TO BOTTOM MENU KEY> to access the Horizontal Options Menu.**

2. **From the Horizontal Options Menu, select the Buffer Menu option.**

   You access the Buffers Menu.

3. **From the Buffers Menu, select the Load Routine option.**

   You see the following prompt in the bottom right-hand corner of the screen:

   ```
   Load Routine:
   ```

4. **At the "Load Routine" prompt, enter the name of the routine you want to load, and press <RETURN>.**

   The Full Screen Editor load the specified routine into the current buffer, overwriting its previous contents.

   Note: Using the Insert Buffer option allows you to insert the specified routine into the current buffer without overwriting its previous contents.

Displaying Multiple Buffers

The Full Screen Editor displays the contents of a buffer in a window. Windows in the Full Screen Editor let you display up to two buffers simultaneously.

You can control the display of your buffers using the Windows Menu.

When displaying two buffers, the Full Screen Editor splits the display screen so that one window occupies the top half of the screen and the other window occupies the remaining area of the screen above the status line.

When two buffers are sharing the screen, you can cut and paste between them.

Editing commands function no differently for two windows than for one. Because the position of the cursor determines the current window status, all editing commands function on the window in which the cursor is located.

Procedure **To display and edit multiple buffers simultaneously:**

1. **Press the <GO TO BOTTOM MENU KEY> to access the Horizontal Options Menu.**
2. **From the Horizontal Options Menu, select the Buffer Menu option.**

   You access the Buffers Menu.
3. **From the Buffers Menu, select the Windows Menu option.**

   You access the Windows Menu.
4. **From the Windows Menu, select the Two Windows option.**

   You see the following prompt in the bottom right-hand corner of the screen:

   ```
   Use Buffer:
   ```
5. **At the "Use Buffer" prompt, enter the name of the buffer you want to display in the second window, and press <RETURN>.**

   The Full Screen Editor splits the screen into two halves, one half for each of the two buffers being displayed.

   Note: You can cut and paste between these two buffers.

Setting a Mark in Your Current Buffer

You can set a mark in your current buffer to define a location within the text.

Setting a mark does not affect the surrounding text in any way. Marks only serve to define locations within the text.

Procedure **To set a mark in the current buffer:**

1. **Position the cursor at the place in your text where you want to set a mark.**
2. **Press the <GO TO BOTTOM MENU> key to access the Horizontal Options Menu.**
3. **From the Horizontal Options Menu, select the Mark Menu option.**

   You access the Mark Menu.
4. **From the Mark Menu, select the Set Mark option.**

   You see the following prompt in the bottom right-hand corner of the screen:

   ```
   Mark Name:
   ```
5. **At the "Mark Name" prompt, enter the name of the mark you want to edit, and press <RETURN>.**

   This sets a mark at the place in your text where the cursor is currently positioned.

   You can relocate the cursor back to this position at any time by selecting the Find Mark option on the Mark Menu and specifying the name of the mark you want to find at the "Goto Mark" prompt.

   You can also delete the mark at any time by selecting the Clear Mark option on the Mark Menu and specifying the name of the mark you want to delete at the "Clear Mark" prompt.

Searching For Text Strings

You can search your buffer for a text string by using the following options on the Horizontal Options Menu:

Option	What It Does
Find String	prompts you to specify a text string and searches for the string from the current cursor position to the end of the buffer
Next Find	searches from the current cursor position to the end of the buffer for the next instance of the same text string
Prev Find	searches from the current cursor position to the top of the buffer for the previous instance of the same text string

ReplacingText Strings

You can use the Replace String option on the Horizontal Options Menu to specify a string of text to search for (at the Replace prompt) and a string of text to replace each instance of the search string (at the With prompt). The Replace String option finds and replaces all instances of the search string from the current cursor position to the end of the buffer.

Using Control Key Commands for Quicker Editing

Many commands in the Full Screen Editor are designated by <CTRL-*letter*> keystroke sequences.

The following table summarizes the keystrokes you can use to issue commands for the Full Screen Editor. None of the information contained here is terminal specific. However, you may find that not all the <CTRL-*letter*> functions listed here are available to you, as they tend to vary with operating system:

Table 4-8: <CTRL-*letter*> Commands

<CTRL-*letter*>	Function
<CTRL-A>	Advances cursor several lines of text.
<CTRL-B>	Backs up cursor to first letter of previous word.
<CTRL-D>	Deletes current character.
<CTRL-E>-<*letter*>	Invokes option from primary menu beginning with specified letter.
<CTRL-F>	Advances cursor to first character of next word.
<CTRL-G>-<CTRL-H>	Returns cursor to first character of current M code line.
<CTRL-G>-<CTRL-J>	Moves cursor to end of last physical line displayed on screen.
<CTRL-G>-<CTRL-K>	Moves cursor to end of current M code line.

Table 4-8: <CTRL-*letter*> Commands (Continued)

<CTRL-*letter*>	**Function**
<CTRL-G>-<CTRL-U>	Moves cursor to beginning of first physical line displayed on screen.
<CTRL-G>-<CTRL-X>	Undoes previous deletion.
<CTRL-H>	Moves cursor one character to left.
<CTRL-J>	Moves cursor down one physical line.
<CTRL-K>	Moves cursor one character to right.
<CTRL-L>	Deletes rest of line from current cursor position.
<CTRL-N>	Next tag.
<CTRL-R>	Moves the cursor back 15 lines, maintaining the same cursor position.
<CTRL-U>	Moves cursor up one physical line.
<CTRL-W>	Deletes to end of current word.
<CTRL-X>	Undoes previous deletion.

Note For a complete listing of all keyboard-specific Full Screen Editor action commands for each terminal type supported by InterSystems, see the Appendix to this manual.

Getting Help

You may press the <EXPLAIN> key at any time while the cursor is located in the Edit Field to access the Full Screen Editor Help Menu. The Help Menu is a horizontal options menu that appears at the bottom of the screen.

Use the <LEFT ARROW> and <RIGHT ARROW> keys to navigate the menu, and press <RETURN> to select an option.

The table below lists and describes the options on the Help Menu:

Table 4-9: Options on Help Menu

Option	Function
General	This option displays a list of all control sequence commands currently defined for the Full Screen Editor.
Keys	This option is not currently implemented.
Options	This option is not currently implemented.
Action	This option lets you select an action on which you would like more information. When you select this option, you see the following prompt in the bottom right-hand corner of the screen: `Action:` At the "Action" prompt, specify the name of an action on which you would like more information, and press <RETURN>. The system displays a help window that contains a description of the specified action. You may also perform lookups of actions at the "Action" prompt. Enter the first letter or sequence of letters for an action and press <RETURN> to see a lookup box that lists all actions with matching names.

Exiting the Full Screen Editor

When finish editing a routine in the Full Screen Editor, you may press the <PREVIOUS> key to access the Full Screen Editor Save Menu. The Save Menu lets you save edits to a routine, compile a routine, rename a routine, and exit the Full Screen Editor.

You may press the <PREVIOUS> key at any time while the cursor is located in the Edit Field. The Save Menu is a horizontal options menu that appears at the bottom of the screen. Use the <LEFT ARROW> and <RIGHT ARROW> keys to navigate the Save Menu, and press <RETURN> to select an option.

The table below lists and describes the options on the Save Menu:

Table 4-10: Options on the Save Menu

Option	Function
Quit	Exit the Full Screen Editor without saving or compiling the routine. **Note:** If you select this option while there are unsaved edits made to the routine, the Full Screen Editor displays the following confirmation prompt before allowing you to exit the Editor without saving your edits: `Quit without filing changes?`
Save & Compile	Save and compile routine. This option may disrupt any other process that is running the routine. See the discussion on ZSAVE in the *Open M Programming Guide* to learn the consequences of saving a program that another process is executing.
Only Save	Save but do not compile the contents of the current buffer.
Rename & Save	Save but do not compile the contents of the current buffer as a new routine.

Automatic Syntax Checking

Whenever you compile a macro source or intermediate code routine, the Open M/SQL Compiler automatically syntax-checks the code and records all errors in a sequential error log. The error log displays to the screen upon the completion of the compilation. In the Full Screen Editor, you can display the error log by selecting the Check Errors option on the Primary Menu. The error log displays in its own window located in the lower half of a split screen. This screen split enables you to correct the routine while looking at the error report. To erase the error log window and repaint the screen, type <CTRL-E><1>.

Note The Full Screen Editor does not support syntax checking for Open M/SQL systems running on non-ISM host M implementations.

Automatic Date and Time Stamps

The Full Screen Editor can automatically stamp the date and time in the form of a comment on the first line of intermediate code routines whenever changes are saved.

To enable this feature as the system-wide default behavior, the System Manager must issue the following command from the System Manager's directory:

```
> set ^%rde("MARK")=1
```

CHAPTER

5

Developing Macro Source Routines

InterSystems recommends that you write all routines (even pure M routines) at the top level — the macro source level. In order to create a macro source code routine, you must use the Full Screen Editor.

This chapter describes how to develop macro source routines. Specifically, it covers the following topics:

- Creating Macro Source Routines
- The Open M/SQL Macro Preprocessor
- Summary of Macro Preprocessor Commands
- Summary of Macro Preprocessor Functions

Creating Macro Source Routines

To create and edit both macro source routines and include files in Open M/SQL, you must use the Full Screen Editor.

Note You can also create intermediate code routines using the Full Screen Editor, but InterSystems recommends creating routines at the macro source level.

Macro source code permits the use of macros and embedded SQL statements using a combination of ANSI-Standard M syntax, special macro preprocessor commands, and ANSI-Standard SQL. Macro source routines can also refer to include files, which are useful for standardizing the behavior of a set of programs and for customizing a single source for different environments.

Macro source code can use preprocessor commands and keywords to do all of the following:

- Provide names for constants, expressions, and other arbitrary text, which can be replaced at compile time without sacrificing run-time performance.
- Conditionally include lines of code.
- Include macro source code from named include files.
- Tailor a single macro source routine to different environments.
- Execute SQL statements as part of a program.

Compiling Macro Source Routines

When you compile macro source routines, the Compiler works in two phases. First, the Compiler converts the macro source routine into pure M code, called intermediate code. This level of compilation is called the macro preprocessor phase. Subsequently, the Compiler compiles the intermediate code routine into executable code, called object code.

To compile a macro source routine, you may select the “Save and Compile” option of the Full Screen Editor after editing a macro source routine, or you may compile the routine directly from the M programmer prompt using the %urcomp utility.

Macro Source Routines and Include Files

Macro source code can use include files.

Include files contain definitions that are used in the preprocessor phase of compilation to expand macros and determine whether optional lines of code should be included. They can also be used to include a common block of code in several routines, saving the overhead of calls to a common subroutine.

The table below summarizes the differences between include files and a macro source routines:

Table 5-1: Macro Source Routines versus Include Files

Macro Source Routine	Include File
Created using Full Screen Editor	Created using Full Screen Editor
Named with suffix .MAC	Named with suffix .INC
Compiler produces intermediate code from the macro source routine, then translates intermediate code into executable object code routines	Cannot invoke Compiler directly; must be referenced in a macro source routine
Comments in source are included in the intermediate file	Comments in include file are not included in the intermediate file unless specified by the #show command

The Open M/SQL routine management utilities use the filename suffix to distinguish between include files (FILENAME.INC) and macro source routines (FILENAME.MAC). The suffixes may appear in either lower or upper case.

Macro source routine and include file names may include up to 235 alphanumeric characters. They must begin with an alphabetic character. Underscores are not allowed. Case is significant.

Macro Source Routines Are Portable Across M Implementations

Macro source routines are completely portable across Open M/SQL systems running on any of the following M implementations:

- ISM
- DTM
- DSM
- MSM

You may create macro source routines in Open M/SQL on any of these M implementations, compile the routine into intermediate code, and then run it on the same or any other M implementation.

The Open M/SQL Macro Preprocessor

The Open M/SQL macro preprocessor recognizes three kinds of constructs, as shown in the following table:

Construct	Symbol
Preprocessor Commands	#
Preprocessor Functions	## or &
Macro References	$$$

Macro Preprocessor Commands

A preprocessor command can prompt the preprocessor to execute any of the following actions:

- Define or undefine a macro
- Include the text of an include file
- Conditionally process consecutive lines in the current source
- Set a mode of operation for the preprocessor

A preprocessor command must appear on a line by itself, at the left margin or preceded by one or more space or tab characters, as in the following:

```
tag set x=5,y=10
   #include LIBNAME
      quit
```

The following is incorrect:

```
tag set x=5,y=10 #include LIBNAME quit
```

With the exception of the #define command, preprocessor commands can be followed by a comment on the same line.

Macro Preprocessor Functions

A preprocessor function is an expandable construct that can appear anywhere within a line of M code. When you compile the macro source code, the preprocessor line "expands" and is replaced by the designated code. The preprocessor function, &sql (or ##sql), is used to embed SQL statements in an M program.

A preprocessor function can appear anywhere in the text of a macro source routine, as in:

```
set id=5 &sql(FETCH ecurs INTO :name) write !,name quit
```

Even if the function extends across two or more lines, it can be followed by code on its last line, for example:

```
set id=5 &sql(FETCH ecurs
INTO :name) write !,name quit
```

The Compiler does not expand preprocessor functions when they are embedded inside quoted strings, i.e., when enclosed in double quote characters (""), or in comments.

Macro References

A reference to a previously defined macro should consist of the macro name preceded by 3 dollar signs ($$$) and optionally followed by one or more arguments in parentheses. When inserted, a macro reference is replaced by the definition of the macro as established by a previously specified macro define (#define) statement. If the macro has been defined to take arguments, argument substitution is performed to generate the text value of the macro reference.

A macro reference must be preceded and followed by punctuation characters or the beginning or end of the macro source line. The macro definition:

```
#define DATE $ZD($H,2)
```

assigns the value "$ZD($H,2)" to the macro "DATE". Thus, a subsequent occurrence of the macro source text, such as:

```
w !,"The date is ",$$$DATE
```

is replaced by:

```
w !,"The date is ",$ZD($H,2)
```

The following macro reference:

```
xyz$$$version
```

is invalid because it immediately follows an alphanumeric character.

The source text:

```
$$$versionxyz
```

is interpreted as a reference to the macro named “versionxyz”. Since a macro reference cannot be adjacent to an alphanumeric character, you cannot use a macro to generate part of an identifier name.

The Compiler does not expand macro references and preprocessor functions when they are embedded inside quoted strings, i.e., when enclosed in double quote characters (“”), or in comments.

References to undefined macros will produce error messages at compile time. The “$$$” preceding the macro name is left intact.

Macro source lines that, after expansion, do not contain any characters other than space and tab are omitted from the intermediate routine.

Summary of Macro Preprocessor Commands

The following table lists and briefly describes the commands supported by the macro preprocessor:

Table 5-2: Macro Preprocessor Commands

Preprocessor Command	What It Does
#define	Define a macro.
#undef	Remove a macro definition.
#include	Include macro source text from a specified, previously created include file.
#show	Include comments from include files.
#noshow	Don't include comments from include files. (Default)
#if	Conditionally include the following macro source text if an expression is true.
#ifdef	Conditionally include macro source text if a specified macro is defined.
#ifundef (#ifndef)	Conditionally include macro source text if a specified macro is not defined.
#else	Include macro source text if the previous #if, #ifdef, or #ifundef failed.
#elseif (#elif)	Include macro source text if the previous #if, #ifdef, or #ifundef failed and an expression is true.
#endif	Terminate conditional text.
#;	Define single-line, macro-only comment lines

#define and #undef

The #define statement can appear in several forms and the #undef statement in just one form:

```
#define MACRONAME
#define MACRONAME VALUE
#define MACRONAME(PARAMETERS) VALUE

#undef MACRONAME
```

where MACRONAME is a valid macro name (without the $$$) and VALUE, separated from MACRONAME by at least one space, consists of the rest of the line. VALUE can be any arbitrary text. The macro preprocessor strips leading and trailing spaces from VALUE. If the line ends in a comment, the comment is included in the macro value.

#define MACRONAME

The first form of #define causes MACRONAME to be defined with a null value. This is useful primarily in combination with the #ifdef and #ifundef commands (described below) that test whether or not a macro is defined.

#define MACRONAME VALUE

The second form of #define causes MACRONAME to be defined with the value of VALUE, for example:

```
#define release 3
```

A later reference to this macro, such as:

```
go:currel<$$$release oldrel
```

expands into:

```
go:currel<3 oldrel
```

Another example is the definition:

```
#define var "^abc(qsub"
```

for which the following references:

```
set x=$$$var_",n)"
set y=$$$var_")"
```

expand into:

```
set x=^abc(qsub,n)
set y=^abc(qsub)
```

#define MACRONAME() VALUE

The third form of #define defines a macro that takes arguments, also called a function macro. The argument(s) may be one or more alphanumeric strings, each beginning with percent signs and separated by commas. Each of the parameter strings can occur one or more times in VALUE, indicating a substitution.

A reference to a function macro takes the form $$$MACRONAME(), where the argument(s) are any character strings separated by commas. Commas and right parentheses can be passed as macro arguments only if they are part of quoted strings. At expansion time, the macro preprocessor expands any $$$MACRONAME references inside the parentheses, then substitutes the arguments for the corresponding % parameters in the macro's definition.

For example, if the macro “version” is defined as:

```
#define version(%a,%b) %a_".0"_%b
```

then the reference:

```
write “VERSION=”,$$$version($$$release,“subrel”)
```

causes the macro preprocessor to replace “$$$release” with “3” (defined in the example above) and then substitute the arguments “3” and “subrel” into the text to produce:

```
write “VERSION=”,3_“.0”_“subrel”
```

When a macro is defined to accept arguments, you must reference it with the correct number of arguments; otherwise, the macro preprocessor generates an error. If a macro is defined without arguments, you may never include parentheses in a reference to that macro. For example, the following source text:

```
#define foo “precise”
$$$foo()_“ly”
```

expands into:

```
precise()ly
```

In this case, the macro preprocessor will issue a warning message about the null parentheses.

#undef MACRONAME

The #undef statement causes MACRONAME to have an undefined value. The distinction between a null value and an undefined value is important to the #ifdef and #ifundef commands.

Nested Expansion

Nested expansion occurs when one expandable property contains a second expandable property within its expanded evaluation. Macro definitions, include file names, and embedded SQL text can all contain $$$MACRO references. When this happens, the macro preprocessor must resolve nested expansion.

A macro reference is always expanded at the time that it is used to produce actual text. For example, when the macro preprocessor encounters the #include command:

```
#include $$$system
```

it must evaluate the macro reference $$$system to decide which include file to use.

Macro references inside embedded SQL are expanded before the SQL text itself is evaluated. For more information on using macro references in embedded SQL, refer to the section entitled “Referencing Macros in Embedded SQL” in Chapter 8, *Embedded SQL*.

When a macro reference is expanded, the macro preprocessor scans the result for additional macro references. If it finds another macro reference, the macro preprocessor expands it, then again scans the result for further macro references, and so forth until no macro references remain.

If a macro reference is located inside a #define statement, it is not evaluated until the macro being defined is referenced. For example, the following define statements:

```
#define release 3
#define version(%a) $$$release_“.0”_%a
```

cause the function macro “version” to be defined with the value:

```
$$$release_“.0”_%a
```

where %a is the macro’s argument. The macro preprocessor expands the reference to $$$release only when the macro “version” is referenced. The source text:

```
write $$$version(“subrel”)
```

expands into:

```
write $$$release_“.0”_“subrel”
```

and finally into:

```
write 3_“.0”_“subrel”
```

If a #define statement later redefines “release” to be 4, then a subsequent occurrence of the statement “write $$$version(subrel)” in the macro source will expand into:

```
write 4_“.0”_“subrel”
```

#ifdef, #ifundef, #if, #else, #elseif, and #endif

The #ifdef preprocessor command includes a block of source text only if the specified macro name has a defined value.

The #ifundef command (also abbreviated #ifndef for compatibility with the C programming language) has the opposite meaning. It includes a block of source text if the specified macro name does not have a defined value.

The #if command includes a block of source text if the specified expression evaluates to true (i.e., a non-zero, numeric value). The expression consists of M macro source that, after expansion, produces an M expression. Since this expression is evaluated at compile time (not at run time), any references to local variables or extrinsic functions are resolved in the environment in which the program is compiled. Since the compilation environment is difficult to predict, you should use caution when referencing local variables and extrinsic functions inside #if expressions.

The #else command specifies a block of source text to be included if the previous #ifdef, #ifundef, #if, or #elseif command was not satisfied.

The #elseif command (also abbreviated #elif) includes a block of source text if the previous #ifdef, #ifundef, #if, or #elseif command was not satisfied and if the specified expression evaluates to true.

These statements have four syntax options, as described in the following sections.

Syntax 1

```
#if EXPRESSION
   (SOURCE TO INCLUDE IF EXPRESSION IS TRUE)
#else                                              ;optional
   (SOURCE TO INCLUDE IF EXPRESSION IS FALSE)  ;optional
#endif
```

For example:

```
#if $$$version<$$$LatestVersion
    do convert($$$version)
#endif
```

Syntax 2

```
#if EXPRESSION A
   (SOURCE TO INCLUDE IF EXPRESSION A IS TRUE)
#elseif EXPRESSION B                              ;optional
   (SOURCE TO INCLUDE IF EXPRESSION A IS FALSE AND
    EXPRESSION B IS TRUE)                         ;optional
#elseif EXPRESSION C                              ;optional
   (SOURCE TO INCLUDE IF EXPRESSIONs A & B ARE
    FALSE AND EXPRESSION C IS TRUE)               ;optional
#endif
```

For example:

```
#if $extract($$$application)="F"
   #include FINANCIAL
#elseif $extract($$$application)="S"
   #include SALES
#elseif $extract($$$application)="M"
   #include MARKETING
#endif
```

Syntax 3

```
#ifdef MACRONAME
   (SOURCE TO INCLUDE IF MACRO DEFINED)
#else                                             ;optional
   (SOURCE TO INCLUDE IF MACRO NOT DEFINED)       ;optional
#endif
```

For example:

```
#ifdef debug
   if dbmode="halt" do ^dbhalt
#else
   if dbmode="go" do ^dggo
#endif
```

Syntax 4

```
#ifundef MACRONAME
   (SOURCE TO INCLUDE IF MACRO NOT DEFINED)
#else                                                ;optional
   (SOURCE TO INCLUDE IF MACRO DEFINED)              ;optional
#endif
```

For example:

```
#ifundef TERMTYPE
   #define BUFSIZE 512
#else
   #define BUFSIZE 1024
   #define DECTYPE 220
#endif
```

Notes

Note the following:

- The source code conditionally included by #if, #ifdef, and #ifundef may include preprocessor commands. As is shown in the above examples, #if's, #ifdef's, and #ifundef's may be nested.
- The names of macros referenced inside #if expressions must be prefaced by $$$.
- Indentation of preprocessor commands, as is used in the above examples, is optional. It is used here to assist readability.

#include

The preprocessor command:

```
#include FILENAME.INC
```

causes the macro preprocessor to treat the contents of the include file FILENAME.INC in the current directory as the next part of the macro source code. The include file can contain any kind of macro source text, including nested #includes.

The differences between an include file and a macro source routine are:

- Include files have the suffix .INC. Macro source routines have the suffix .MAC.
- You cannot directly invoke the Compiler on an include file.
- Whereas comments in a macro source routine are included in the intermediate code routine, comments in an include file are not, unless otherwise specified by the #show command.

When you compile a macro source routine that references an include file, the macro preprocessor replaces the #include command with the text of the include file. If the include file includes macro definitions, the macro preprocessor expands those definitions as it encounters them. You may also have an #include command within an include file. This causes nested inclusion.

#show, and #noshow

The #show command instructs the macro preprocessor to preserve all subsequent comments in an include file in the intermediate code routine.

The #noshow command restores the default condition of not preserving comments in an include file in the intermediate code routine.

The macro preprocessor always preserves comments in the macro source code in the intermediate code.

Advantages of Using Include Files

Include files are useful for standardizing the behavior of a set of programs and for customizing a single source file for different environments.

More specifically, some of the advantages to using include files are:

- Processing a single macro source with different include files is a way to customize an application for different environments.
- Using macro names defined in a commonly shared include file to stand for constant values ($$$MAXSIZE instead of 511, or $$$REPEATSW(10,251) instead of $p(^mdd(1,10,2,251,3),$c(1),6)) reduces the likelihood of errors in which two cooperating programs fail to use the same value, or look in different places for an item of data. It also does not suffer the performance penalty of using a variable in place of the constant.
- If several programs need to execute the same block of M code, without the added overhead of calls to a centralized routine, you can put that code in an include file and have each macro source routine reference that include file. Doing so ensures that each program will execute exactly the same code.

For example, suppose you want to customize a single set of macro source routines for environments A, B, and C by having them all share the include file "FILEA.INC", "FILEB.INC", or "FILEC.INC".

You begin each macro source routine with the line:

```
#include FILECHOICE
```

The include file FILECHOICE.INC consists of the single line:

```
#include FILENAME
```

where "FILENAME" is "FILEA", "FILEB", or "FILEC". Each of the include files "FILEA.INC", "FILEB.INC", and "FILEC.INC" contains a set of #define commands used to customize the environment for A, B, or C. To tailor all routines for a particular environment, you may edit the "FILENAME" value in "FILECHOICE.INC" and recompile all the macro source routines.

Indicating Comment Lines

You may use the #; preprocessor command to provide a single-line of comment text within a macro source routine. The comment line appears in the macro source code but is suppressed by the macro preprocessor from the intermediate code.

For example:

```
#; This is an old version
```

The pound sign (#) and the semicolon (;) may be separated by any number of spaces.

When you use this preprocessor command, your comment text must not exceed one line.

Summary of Macro Preprocessor Functions

The following table lists and briefly describes the functions supported by the macro preprocessor:

Table 5-3: Macro Preprocessor Functions

Preprocessor Function	What It Does
##sql &sql	Delimits embedded SQL code. Either symbol is acceptable.
##vendor	InterSystems-specific preprocessor function used to make macro source routines portable among Open M/SQL-supported host M systems (ISM, DSM, DTM, and Micronetics' MSM).

&sql(...)

To embed an SQL statement within an M macro source routine, you must use the Open M/SQL preprocessor syntax &sql(...) or ##sql(...). Either syntax is acceptable.

The embedded SQL statement begins after the prefix &sql(and concludes with the matching right parenthesis. Embedded SQL can stretch across multiple lines or can occupy just a portion of a single line. However, you may not split SQL keywords or tokens across lines.

The following is an example of a single-line embedded SQL statement where SQL code is intermixed with M code on the same line:

```
for i=1:1 &sql(fetch c into :x,:y) quit:SQLCODE=10020 do
output
```

The following is an example of a multi-line embedded SQL statement:

```
&sql(DECLARE xcurs CURSOR FOR SELECT
Name,Age FROM Patients WHERE Age <12 AND
Ward="4D")
```

SQL statements, like the rest of the macro source code, can contain $$$MACRO references. The macro preprocessor expands and replaces macro references inside embedded SQL statements before evaluating the SQL statements themselves.

The macro preprocessor reports an error if it reaches the end of the macro source routine without encountering the right parenthesis to match "&sql(".

For a description of how M routines can use embedded SQL to access an Open M/SQL database, see Chapter 8, *Embedded SQL*.

##vendor

##vendor is an InterSystems-specific preprocessor function used to make Open M/SQL routines portable among the host M systems supported by Open M/SQL. The Open M/SQL-supported host M systems include:

- ISM
- DTM
- DSM
- Micronetics' MSM

You can observe the use of the ##vendor(...) syntax throughout the Open M/SQL system-generated code. For example, you can see it in default-generated Internal-to-External and External-to-Internal Conversion Code, Validation Code, field length code, and help message code.

At the macro source level, you may use ##vendor to delimit vendor-specific M constructs, thus producing vendor-independent code. This allows the macro preprocessor to compile the routine into intermediate code for the specified target M system.

InterSystems reserves ##vendor for its own use.

CHAPTER

6

Routine Handling and Maintenance

This chapter presents an overview of the Open M/SQL routine environment and discusses topics related to routine handling and maintenance, including information on referencing routines by name, extension, and version as well as instructions for copying, compiling, and backing up routines.

Specifically, it covers the following topics:

- Routine Environment
- Routine Names, Extensions, and Version Numbers
- Using Wildcard Syntax to Specify Routines
- Referencing Routines in Other Directories
- Routine Sets
- Compiling Routines
- Backing Up Routines
- Deleting Routines
- Routine Copying and Compiling Synchronization

Routine Environment

In Open M/SQL, you can create routines at two levels:

- Macro source
- Intermediate code

The macro source level permits the use of macros and embedded SQL statements using a combination of ANSI-Standard M syntax, special macro preprocessor commands, and ANSI-Standard SQL. Macro source routines can refer to include files, which are useful for standardizing the behavior of a set of programs and for customizing a single source for different environments.

You can also write pure M routines (routines that do not include embedded SQL or any macro preprocessor statements) at the macro source level.

When macro source code is compiled, it is first converted into M code with embedded SQL, called intermediate code. This level of compilation is called the preprocessor phase. Intermediate code routines are subsequently compiled into executable object code.

Writing Routines

InterSystems recommends that you create all routines (even pure M routines) at the top level — the macro source level. In order to create a macro source code routine, you must use the Full Screen Editor. Where macro source code exists, you should always compile from the macro source level. When macro source code is compiled, it produces both intermediate code and object code. When a macro source routine contains embedded SQL or refers to an include file, or both, intermediate and object code must always be regenerated from the macro level. The macro source level allows you to save backup copies of routines. For more information on writing routines at the macro source level, refer to Chapter 5, *Developing Macro Source Routines*.

You may also create routines directly at the intermediate code level. For information on developing and editing intermediate code routines on an ISM system, refer to the *Open M/SQL M Programming Guide*.

Converting Intermediate Code to Macro Source Code

Open M/SQL allows a mixed environment in which some routines have macro source versions and others have only intermediate code versions. You can copy intermediate code routines to the macro source level using the %urcopy utility, provided the intermediate code routines do not include embedded SQL. This is a particularly useful feature if you are converting pure M applications to applications that make use of macros, SQL, and other relational database features.

Routine Names, Extensions, and Version Numbers

When using the Full Screen Editor or any of the routine management utilities provided by InterSystems, you may refer to routines and include files not only by name but also by extension and version number.

The complete syntax for routine identification is:

```
NAME.EXTENSION.VERSION
```

For example:

```
ROU.MAC.1
ROU.MAC.2
ABC.INC.1
```

Sometimes you may use the remote directory syntax to specify a routine that resides in a directory other than the current directory, for example:

```
["DIR"]ROU.MAC.2
```

Routine Names Must Be Unique

A routine may have the same name as an include file, but no two routines in a single directory may have the same name, and no two include files may have the same name.

Case Sensitivity

Routine and include file names are case sensitive; thus, "ABC.INT" is not the same as "abc.INT".

Routine extensions are not case sensitive; thus, "ABC.INT" is the same as "ABC.int".

Routine Extensions

The following table lists the routine extensions:

Extension	Meaning
.MAC	For macro source routines
.INT	For intermediate code routines
.INC	For include files
.OBJ	For object code routines

For example, the routine “ROU” might have the following names:

Specification	Meaning
ROU.MAC	Macro source level for ROU
ROU.INT	Intermediate level for ROU
ROU.OBJ	The object code level for ROU

You can also create/edit include files using the Full Screen Editor as well as manipulate them using the routine management utilities. To identify an include files, you must specify the extension .INC, as in:

```
ABC.INC
```

which specifies the include file “ABC”.

When No Extension Is Specified

If you do not specify a file extension for a routine, Open M/SQL assigns the .MAC suffix by default. If no .MAC routine exists, Open M/SQL assigns the .INT suffix.

This allows M programmers who do not make use of the macro source level but rather write routines that consist only of pure M code to use the routine management utilities without adapting their routines or programming methods.

In a mixed programming environment where some routines have a macro source level and others do not, programmers can still use the routine utilities without specifying extensions. In this case, when routines have a macro source level, processing takes place at that level. When only intermediate code exists, processing begins at the intermediate level.

Version Numbers

Macro source routines and include files can have up to 9 backup versions.

Intermediate and object code routines cannot have backup versions. Therefore, you never need to specify version numbers for .INT or .OBJ routines; their version number is always 1 implicitly.

The table below shows how you refer to multiple versions of a macro source routine called “ROU”:

Specification	Meaning
ROU.MAC.1	Current version
ROU.MAC.2	1st backup
ROU.MAC.3	2nd backup

etc., up to a maximum of ROU.MAC.9.

The table below shows how you refer to multiple versions of an include file called “ABC”:

Specification	Meaning
ABC.INC.1	Current version
ABC.INC.2	1st backup
ABC.INC.3	2nd backup

etc., up to a maximum of .INC.9.

You must explicitly specify the version number only when referring to versions other than the current version (version 1). Thus, the following specifications are equivalent:

```
.MAC    =   .MAC.1
```

and

```
.INC    =   .INC.1
```

Using Wildcard Symbols to Specify Routines

InterSystems provides a series of wildcard symbols to assist you in specifying sets of routines to be acted on by the routine management utilities. You may use these wildcard symbols when specifying routines at the Routine(s): prompt.

Wildcards for Routine Names

When specifying routine names, you may use any of the special character ("wildcard") symbols described in the table below:

Table 6-1: Wildcard Symbols for Use When Specifying Routine Names

Wildcard Symbol	Meaning
*	Signifies zero or more characters. For example: * means all names AB*D means all names that start with AB and end with D
?	Signifies one and only one wildcard character. For example: A?C means all names that start with A, end with C, and have exactly one character in between.
:	Signifies a range. For example: AB:D means all names from AB through D inclusive.
‘	Signifies exclusion from a previously specified list. For example: A* ‘ABC means all names that start with A except for ABC. Similarly, A* ‘AB:AD ‘AF* means all names that start with A except for those in the range AB through AD inclusively and those that start with AF.
=	In utilities that use the two-column format (e.g., %urcopy, where routines in the first column are copied to those in the second), the equals sign (=) can be used to signify the same routine name as specified in the "From" column. If no directory is specified, Open M/SQL assumes the current directory. If no extension is specified, Open M/SQL assumes ".MAC".
=.EXT	The equals sign followed by an extension (=.EXT) can be used to signify the same routine name as specified before with the specified extension.
=:.	The equals sign followed by a colon and period (=:.) can be used to signify the same routine and extension as in the "From" column.

Note Type a question mark (?) at the Routine(s) prompt to view help text with information about the wildcard symbols. Type "?L" to see a list of the routines you have chosen so far.

Wildcards for Extensions

You may also use the asterisk symbol (*) in the file extension identifier. In this case, it means all extensions for the specified routine name(s).

For example:

```
ROU.*
```

expands to:

```
ROU.MAC, ROU.INT, ROU.INC, and ROU.OBJ
```

More specifically, it expands to:

```
ROU.MAC.1, ROU.INT, ROU.INC.1, and ROU.OBJ
```

Wildcards for Version Numbers

It is necessary to specify a version number only when the version you are referencing is not the current version (version 1).

When no version number is specified, the version is implicitly .1.

You may use the asterisk symbol (*) in the version identifier. In this case, it means all versions for the specified routine(s).

The table below shows several examples of the use of the asterisk symbol in the version identifier:

Specification	Meaning
ROU.MAC.*	Means ROU.MAC.1, ROU.MAC.2, etc.
ROU.OBJ.*	Means ROU.OBJ.1 (there are no backups)
ROU.INC.*	Means ROU.INC.1, ROU.INC.2, etc.
ROU.*.*	Means all versions of all extensions of routine ROU
..*	Means all versions of all extensions of all routines

Referencing Routines in Other Directories

Open M/SQL's routine management utilities allow you to reference routines located in directories other than the current directory, including directories on other computers, where applicable.

To specify routines located in a directory other than your current directory, you may use remote directory syntax at the Routine(s): prompt, as shown below:

```
[directory,directory_set_name]routine.extension.version
```

where *directory* is the name of the target directory and *directory_set_name* is the name given to the target computer in your M/NET networking configuration.

If your target directory is the current directory, you do not need to specify remote directory syntax at all. If your target directory is another directory on the same computer, you may ignore the *directory_set_name* parameter.

Note If you are running Open M/SQL on a non-ISM host M system, the remote directory syntax may be different. Consult your system guide for this information.

For example:

```
Routine(s): ["DIR","SYS"]ROU.MAC
Routine(s): ["DIR2"]ABC.INT
```

This example selects the .MAC version of routine "ROU" in directory "DIR", directory set "SYS" and the .INT version of routine "ABC" in directory "DIR2" on the current computer.

To avoid repeatedly retyping remote directory information where a list of routines is required, you may use the following syntax to reference the last explicitly specified directory:

```
[^]
```

The following example selects routines "AAA", "BBB", and "CCC" in directory "DIR$SYSX":

```
Routine(s): ["DIR$SYSX"]AAA
Routine(s): [^]BBB
Routine(s): [^]CCC
```

For routine management utilities that use two-column format (e.g., %urcopy, where routines from column one are copied to those in column two), the [^] syntax is column-specific. In the following example, routine "AAA" in "DIR1" of system "SYS" is copied to routine "BBB" in the current directory, then routine

"XXX" also in directory "DIR" of system "SYS" is copied to "YYY" in the current directory:

```
Copy Routine(s): ["DIR1","SYS"]AAA     To: BBB
Copy Routine(s): [^]XXX                To: YYY
```

Restrictions on Using Remote Directory Syntax

Use of remote directory syntax has the following restrictions:

1. The routine management utilities do not permit you to alter the contents of a directory other than the current directory.
2. The utilities %urchange, %urcomp, and %urdel do not support remote directory syntax.
3. The %urcopy utility lets you copy routines from a remote directory into the current directory, but you cannot copy routines from the current directory into a remote directory.

Routine Sets

The Open M/SQL Routine Set facility allows you to create a list of routines under one name and reference that routine set name at the Routine(s): prompt for any of the routine management utilities. This feature is useful when you have a group of routines on which you commonly perform a particular function, such as compiling all the routines in a particular application.

Creating a Routine Set

You may create a routine set at the Routine(s): prompt of any of the routine management utilities.

Procedure **To create a routine set:**

1. **At the Routine(s) prompt for any of the routine management utilities, specify all the routines you want to include in the routine set, either by naming them explicitly or by using the wildcard syntax.**
2. **When you have named all the routines you want to include, type ".F" at the next appearance of the Routine(s): prompt, and press <RETURN>.**

   The system prompts you to provide a name and description for the routine set, as shown below:

```
>d ^%urdir
Routine(s):  test*.mac
Routine(s):  .F
  File as Routine Set: TESTSET
   With Description: Routines for testing
  OK to File? Yes= <RETURN>
Filing ... done
```

   Routine set names are case sensitive, so "ROUTINESET" is not the same as "routineset".

Using a Routine Set

To use a pre-defined routine set, type the routine set name preceded by the "@" character at the Routine(s): prompt, as shown in the example below:

```
>d ^%urdir
Routine(s):  @TESTSET

                        --.MAC--
  test1.mac  test2.mac  test3.mac  test4.mac  test5.mac
```

Compiling Routines

You may use the %urcomp utility to compile either the macro source or intermediate code level of a routine.

When you invoke %urcomp on a macro source routine, Open M/SQL compiles the routine in two phases. First, the macro preprocessor phase of the Compiler produces intermediate code, then the main Compiler produces object code.

If you invoke %urcomp on intermediate code routines, the main Compiler directly produces object code.

Some intermediate code routines cannot be compiled. When the Compiler cannot compile a routine, it issues an abort compilation message and does not modify the routine. The following two conditions can cause this to happen:

1. Source lines are missing from the intermediate code level of ^ROUTINE.
2. The intermediate code contains embedded SQL.

In either case, you must compile the routine at the macro source level.

Most M-level and SQL-level syntax errors do not cause the compiling process to abort.

Other ways to compile a routine include:

1. You may compile a routine as a result of executing the utilities %urchange and %urcopy. These utilities prompt you to specify whether or not you want to compile the specified routines.
2. You may elect to compile a routine from within the Full Screen Editor.

Backing Up Routines

The Full Screen Editor and the utilities %urchange, %urcomp, and %urcopy all provide the option of producing backup versions of macro source routines and include files.

You can use the %urverma utility on a per-directory basis to set the maximum number of versions to be maintained for macro source routines and include files. The default number of versions is four (one current version and three backups). The maximum number of versions that Open M/SQL can maintain is nine (one current version and eight backups).

Open M/SQL maintains backups for macro source routines and include files only. It does not maintain backups for intermediate code or object code routines; their version numbers are always .1.

How Backups are Shuffled and Renumbered

Whenever Open M/SQL generates a backup version of a macro source routine or include file, it shuffles the existing backups down. When the number of backups exceeds the maximum number of versions to be maintained, which you may set using the %urverma utility, Open M/SQL deletes the last backup on the list. For example, suppose the following backup versions of the macro source routine "ABC" are maintained:

Routine Name	Version
ABC.MAC.2	First backup
ABC.MAC.3	Second backup
ABC.MAC.4	Third backup

When you modify the current version of the routine and then save it, Open M/SQL generates a new backup, and the other backups shuffle down, as shown below:

Original Name	New Name
ABC.MAC	ABC.MAC.2
ABC.MAC.2	ABC.MAC.3
ABC.MAC.3	ABC.MAC.4
ABC.MAC.4	Deleted

You can use the utilities %urpurge and %urdel to delete old backup copies of macro source routines and include files.

The Full Screen Editor Generates Backups When You Save

When editing a macro source routine or include file in the Full Screen Editor, the Editor automatically generates a backup copy of the routine or file and shuffles down its existing backups (in accordance with the per-directory backup maximum) whenever you elect to save and compile your edits. For example, when you save and compile the macro source routine "ABC.MAC" after editing it in the Full Screen Editor, the previous current version becomes "ABC.MAC.2", and the version you are editing becomes the current version ("ABC.MAC"). Any older versions of the routine ("ABC.MAC.3" through "ABC.MAC.9") are shuffled down and/or out. If you have specified that a maximum of 2 backup versions is to be maintained, the previous "ABC.MAC.1" version is renamed to "ABC.MAC.2", and the previous "ABC.MAC.2" version is deleted.

Restoring a Backup Version to the Current Version

To restore a backup version of a macro source routine or include file to the current version, you may use the %urcopy utility. To do this, you copy *from* the backup version you want to restore *to* the new version.

For example, copying *from* "ROU.MAC.2" *to* "ROU.MAC.1" effectively restores the first backup, making "ROU.MAC.1" and "ROU.MAC.2" identical.

When you use %urcopy to copy one version of a routine to another version of the same routine and extension, the backups do not shuffle down.

Deleting Routines

To delete routines and include files, you may use either of the two utilities described below.

%urdel

You may use the %urdel utility to delete routines and include files. To use this utility, you must specify a list of the routines and include files with extensions and version numbers that you want to delete. For example, if you want to delete all the versions of the macro source routine and include file “ABC”, you specify:

```
Routine(s): ABC.*.*
```

As another example, you may specify the following:

```
Routine(s): DEF.MAC.*
Routine(s): GHI.MAC.2
```

to delete all .MAC versions of routine “DEF” and macro version 2 (the first backup) of routine “GHI”. Deleting macro version 2 of routine “GHI” causes subsequent macro backups of “GHI” to be shuffled forward, hence the old “GHI.MAC.3” becomes “GHI.MAC.2”.

To delete all macro source and intermediate code routines in the current directory, leaving only object code, type:

```
Routine(s): *.MAC.*
Routine(s): *.INT
```

To delete all levels of all routines and include files in the current directory, type:

```
Routine(s): *.*.*
```

%urpurge

You may use the %urpurge utility to delete some or all backups for macro source routines and include files. This utility prompts you to specify a set of routines and include files that you want to purge. You may only specify routines with .MAC extensions or include files with .INC extensions in the list of routines and include files to purge. The utility also prompts you to specify how many versions of the routine or include file to keep after the purge. The default value is 1, meaning the current version only and no backups.

Routine Copying and Compiling Synchronization

It is the programmer's responsibility to keep the .MAC, .INT, and .OBJ levels of a routine in sync, or out of sync if desired. You keep the different levels of a routine in sync by compiling the routine.

Copying or editing a routine at either the .MAC or .INT level does not automatically result in compiling that routine, although the utilities for copying and editing do give you the option of compiling. Thus, the responsibility for determining when a routine should be compiled is yours; Open M/SQL does not do it automatically and does not attempt to keep compiled routines in sync with source code.

Sometimes you may want to keep the different levels of a routine out of sync. For example, you may want to edit one or more macro source routines for several days and not disturb the .INT and .OBJ levels until all editing is complete.

Although Open M/SQL does permit copying of .MAC file extensions to .INT as well as copying of .INT file extensions to .MAC, InterSystems does not recommend this. .INT files do not always contain all of the information necessary to produce corresponding .MAC files—source lines may be missing or embedded SQL code may exist (for which there is no source code). If a .MAC routine includes preprocessor statements (such as #if statements or macros) or embedded SQL, you should not copy it to the .INT level because it cannot be compiled there.

CHAPTER

7

Open M/SQL Routine Management Utilities

InterSystems provides a set of utilities for examining and manipulating routines and include files. These utilities are collectively known as the Open M/SQL routine management utilities.

This chapter summarizes the routine management utilities, shows how to access them, and then provides a detailed description with examples of how to use each utility.

Specifically, it covers the following topics:

- Summary of Routine Management Utilities
- Accessing the Routine Management Utilities
- %uro
- %uri
- %url
- %urprint
- %urload
- %urdir
- %urchange
- %urcomp
- %urcopy
- %urfind
- %urfand
- %urdel
- %urverma
- %urpurge
- %urset

Summary of Routine Management Utilities

The following table lists and describes the routine management utilities provided by Open M/SQL:

Table 7-1: Open M/SQL Routine Management Utilities

Utility	Meaning	Description
%urprint	Routine Output	Prints selected macro source routine(s) and include file(s) from the current directory to a storage file or to a specified output device.
%urload	Routine Input	Loads macro source routines and include files that have been output to a file by the %urprint utility.
%urdir	Routine Directory	Lists routines in the current directory.
%urchange	Routine Change	Changes all occurrences of specified string(s) in selected routine(s).
%urcomp	Routine Compile	Compiles macro source code into intermediate code, and intermediate code into object code for a specified set of routines.
%urcopy	Routine Copy	Copies macro source routines, itermediate code routines, and include files from any directory into the current directory.
%urfind	Routine Search	Searches through routines for occurrences of one of a specified set of strings.
%urfand	Routine Search	Searches through routines for occurrences of all of a specified set of strings.
%urdel	Routine Delete	Deletes routines and include files from the current directory.
%urverma	Set Maximum Number of Backups	Specifies the maximum number of backup versions maintained in the current directory for macro source routines and include files.
%urpurge	Routine Backup Purge	Deletes backup versions of macro source routines and include files.
%urset	Select a Set of Routines	Creates a set of routines to be used by other utilities for other operations.

Accessing the Routine Management Utilities

There are two ways to access the Open M/SQL routine management utilities:

1. You may call them directly from the M programmer prompt.
2. You may access them as menu options from within Open M/SQL.

Calling the Routine Utilities Directly from M

To call the routine management utilities directly from the M programmer prompt, you issue the following command syntax:

```
do ^%utility_name
```

For example, to call the %urcopy utility from the M programmer prompt, you issue the following:

```
>do ^%urcopy
```

You may use the same syntax to call the routine utilities from within M programs.

Accessing the Routine Utilities from within Open M/SQL

Alternatively, you may access the Open M/SQL routine management utilities by selecting them as options from the Macro Routine Utilities menu, which is a sub-menu of the Developer Utilities menu.

All the routine management utilities listed in the table on the previous page are available as options on this menu, except the following:

- %urprint
- %urload
- %urset

Procedure **To access the routine management utilities via Open M/SQL:**

1. **At the M programmer prompt, type the following command to enter Open M/SQL:**

   ```
   > do ^%msql
   ```

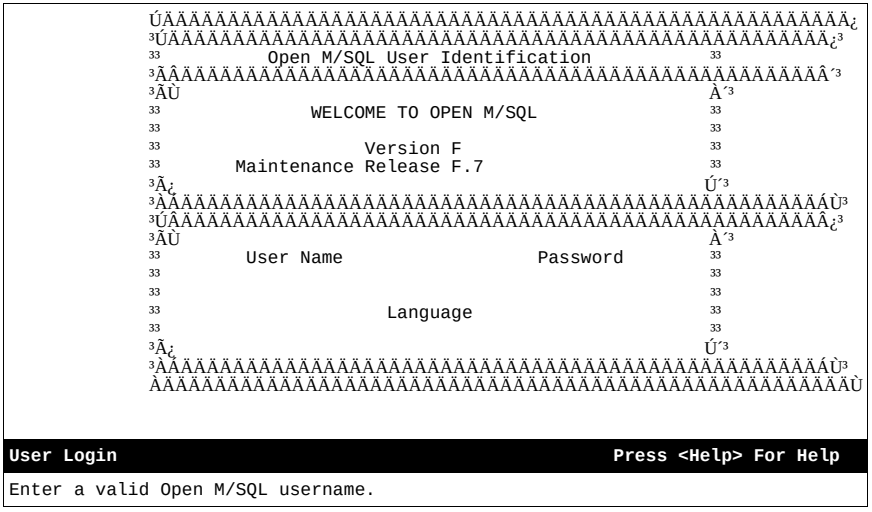
   You see the Terminal Type prompt, as shown below:

   ```
   Terminal Type: VT220 =>
   ```

2. **At the Terminal Type prompt, enter the name of the terminal type you are currently using.**

   You may press <RETURN> to accept the system-wide default terminal type.

You see the Open M/SQL User Identification window, as shown below:



3. **At the UserName field on the Open M/SQL User Identification window, enter your Open M/SQL UserName, and press <RETURN>.**

4. **At the Password field on the Open M/SQL User Identification window, enter the Password for your Open M/SQL UserName, and press <RETURN>.**

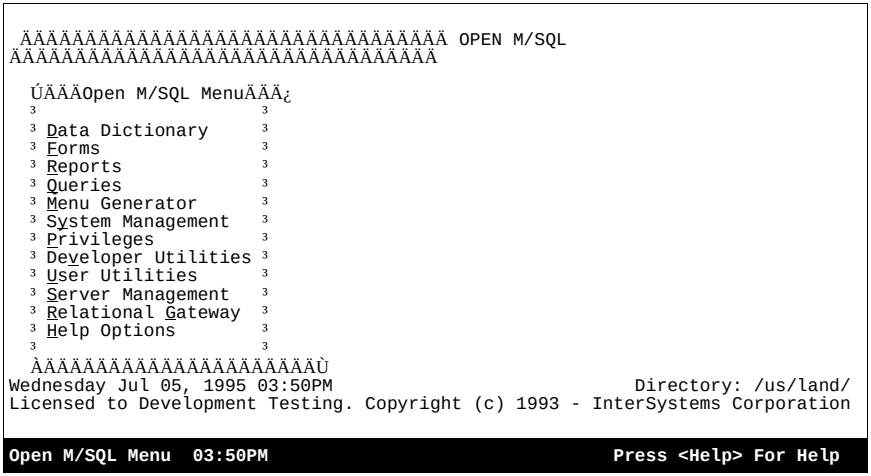
5. **At the Language field on the Open M/SQL User Identification window, enter the language in which you want to run Open M/SQL.**

   To accept the system-wide default run-time language, press <RETURN>.

   To choose a different run-time language, press <CTRL-L> to delete the system-wide default language.

   You may press the <LIST CHOICES> key to see a lookup box that lists the run-time languages supported by Open M/SQL.

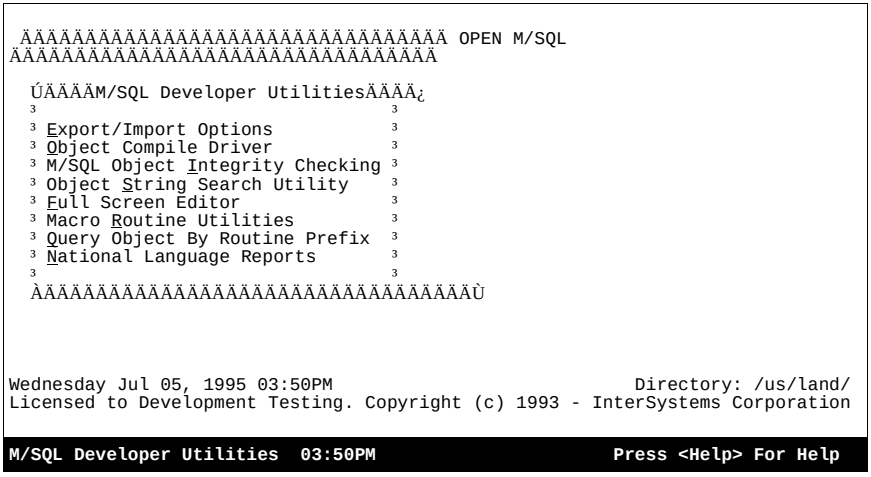
You see the Open M/SQL Main Menu, as shown below:



6. **From the Open M/SQL Main Menu, select the Developer Utilities option.**

   Note: You may type *v* to select this option—it is a mnemonic accelerator.

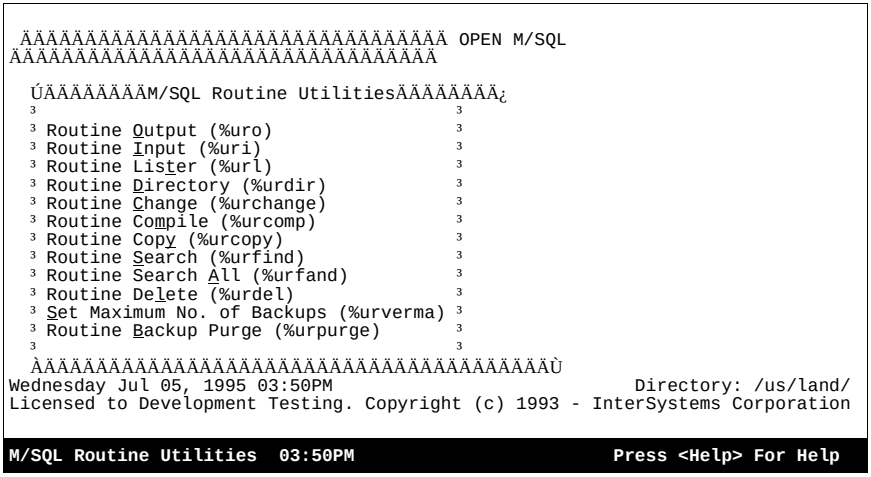
   You see the Developer Utilities menu, as shown below:



7. **From the Developer Utilities menu, select the Macro Routine Utilities option.**

   Note: You may type *r* to select this option—it is a mnemonic accelerator.

You see the M/SQL Routine Utilities menu, as shown below:



8. **From the M/SQL Routine Utilities menu, you may select any option to invoke the corresponding utility.**

   When you exit the utility, you return to the M/SQL Routine Utilities menu.

%urprint

The %urprint utility lets you print selected macro source routine(s) and include file(s) and send the output to a storage file or to a specified output device.

When called, %urprint displays the following prompt:

```
Routine Output
Routine(s):
```

Here you specify the routine(s) you want to output. You may enter a single routine name or a set of routine names with or without extensions. The only valid extensions are .MAC and .INC, which you may specify either in upper or lower case. If you do not specify an extension, %urprint assumes an extension of .MAC.

Note %urprint does not accept intermediate code or object code routines.

Next, %urprint prompts you to enter a description of the output file. Here you may enter some text that describes the file.

Alternatively, you may type the carat character (^) at the "Description:" prompt to add an auto-load header as the first record in the file. An auto-load header is a line of executable M code that, when executed within an Open M/SQL system, causes the contents of the file to be restored automatically. A typical M statement to execute the auto-load header is shown below:

```
OPEN <device>:() USE <device> READ <code> XECUTE <code>
CLOSE <device>
```

Selecting an Output Device

After you specify the routines you want to output, %urprint presents a vendor-specific prompt that asks you to specify an output device. You may send output to any valid device supported by your host M system.

Printing to the Screen

If you specify your current device as the output device, output automatically appears in paged format. At the bottom of each page, the message "More" appears to indicate that the report continues. You may continue the output by pressing <RETURN>, or you may abort the output by typing *Q* or *q* or ^.

Printing to a Printer

If you specify an output device other than your current device, you see the following prompt:

```
Printer Format?
```

If you are directing the output to a printer, you should answer *Yes* at this prompt. Answering *Yes* causes the output to appear one page at a time in a format appropriate for a printer.

Printing to a Storage File

If you specify an output device other than your current device, you see the following prompt:

```
Printer Format?
```

If you are directing the output to a storage file in a directory or on tape so that it may be restored later using the %urload utility, you should answer *No* at this prompt. Answering *No* causes the output to appear in a format useful for backing up the routine, including the appending of a self-loading header record to the beginning of the output file.

Example

In the example below, %urprint outputs the files “ABC.INC” and “ABC.MAC” to a storage file called “/usr/msql/abc.rtn”:

```
> do ^%urprint
Routine Input
Routine(s): ABC.*

Description: ^  (autoload header added)
Output routines to
Device: /usr/msql/abc.rtn  Parameters: "WNS"=> <RETURN>
Printer Format? No => <RETURN>

ABC.INC         ABC.MAC

>
```

You can now restore these files using the %urload utility.

%urload

The %urload utility lets you load macro source routines and include files that have been output to a storage file in a directory or on tape via the %urprint utility.

%urload prompts you to specify an input device, routines to load, and whether or not to compile loaded macro source routines.

When called, %urload displays the following prompt:

```
Enter input device:
Device:
```

Here you must specify the name of a storage device to which routines and include files were output by %urprint.

Routine Input Options

After you specify the input device, %urload displays the following prompt:

```
Routine Input Option: (A)ll ==>
```

This prompt asks you to specify which routines from the input device to load.

Type *?* to see a list of the load options.

The table below lists and describes the load options:

Table 7-2: Load Options for Routine Input Using %urload

Option	Meaning
(**A**)ll	Loads all files on the input device, then prompts you to specify whether or not you want to compile the macro source routines, as follows: `Compile macro source routines? No=>` All is the default option.
(**S**)elect	Prompts you to specify whether or not you want to compile the macro source routines, as follows: `Compile macro source routines? No=>` Then, prompts you file-by-file to specify whether or not to load each file on the input device, as follows: `OK to load? No ==>`
(**E**)nter	Lets you manually specify the list of files to be loaded. When you select this option, you see the following prompt: `Routine(s) To Load:`

Table 7-2: Load Options for Routine Input Using %urload (Continued)

Option	Meaning
(**L**)ist	Lists all files on the input device, then redisplays the "Routine Input Option:" prompt.
(**Q**)uit	Quits %urload.

To select an option, type its first letter, and press <RETURN>.

Compile Macro Source Routines

When you select one of the load options "All", "Select", or "Enter", %urload displays the following prompt:

```
Compile macro source routines? No ==>
```

to determine whether or not you want Open M/SQL to compile all macro source routines as they are loaded.

No is the default response.

Example

In the example below, %urload loads *all* files from the storage file "abc.rtn" into the current directory and compiles them:

```
>do ^%urload
Enter input device:
Device: abc.rtn   Parameters: "R"=> <RETURN>

Routine Input Option: (A)ll ==> <RETURN>
Compile macro source routines? No ==> Yes

ABC.INC                    (loaded)       compiling...
ABC.MAC                    (loaded)       compiling...

>
```

%urload displays a list of all macro source routines and include files from the input device to the screen. It organizes the files in alphabetical order. Beside every file that it loads, it displays the message "(loaded)". Beside every file that it does not load, it displays the message "(skipped)". Beside every macro source routine that it compiles, it displays the message "compiling...".

%urdir

The %urdir utility lists routines in the current directory.

Note On ISM systems the %urdir utility is also known as %RD.

The %urdir utility lets you do all of the following:

- List the routines in another directory.
- Specify any set of .MAC, .INT, .INC, or .OBJ routines to display using the wildcard syntax.
- Select the short or long display form. The short form displays the routine names, extensions, and version numbers. The long form displays, in addition to the short form information, the date and time when routines were last saved, the size in bytes of each routine, and the block(s) that object code routines occupy.
- Specify the selection condition of a range of dates during which the routines were last modified.

First, you see the following prompt:

```
Routine(s)
```

Here you specify the routines that you want to include in the output list.

%urdir supports remote directory syntax, which lets you select routines from a directory other than the current directory. To do this, you must type the name of the source directory inside brackets and within quotation marks [“”], as shown below:

```
Routine(s): ["/us/mgr/tasks"]ABC.MAC
```

After you specify the routine(s) you want to list, %urdir displays the following prompt:

```
Long or Short form (L or S)? S=>
```

Here, you select a display format for outputting the routine list. You may chose between Short Form and Long Form. Short Form lists each routine with its extension and version. Long Form organizes the output into tabular format and provides additional information, including date and time last modified, total number of bytes, and total number of blocks. Short Form is the default choice.

Next, you see the following prompt:

```
Find routines last modified since date:
and on or before date:
```

This is an optional feature that lets you limit the output list to those routines that were modified within a specified date range. Enter a start-of-range date and an end-of-range date, as follows:

```
1/31/95
```

Example 1

In the example below, %urdir uses the Short Form to display of all versions of all macro source routines beginning with the letter '"Y"' in the current directory:

```
>do ^%urdir

Routine(s): Y*.MAC
Routine(s): <RETURN>

Long or Short form (L or S)? S=> <RETURN>
Find routines last modified since date: <RETURN>
          and on or before date: <RETURN>
Display on
Device: <RETURN>            Right margin: 80=> <RETURN>

    Short Listing of Selected Routine/Include Files
                 Directory: /usr/msql
              10 Jul 95  01:24PM   Page 1

YAAA               --  .MAC  --

YAAA              --  .MAC.2  --

YAAA              --  .MAC.3  --

YAAA              --  .MAC.4  --

YBBB               --  .MAC  --

YCCC               --  .MAC  --

YDDD               --  .MAC  --
 >
```

Example 2

In the example below, %urdir uses the Short Form to display all versions of all macro source and intermediate code routines that reside in the current directory, begin with the letter "Y", and were modified between January 21, 1995 and the current date:

```
 >do ^%urdir

 Routine(s): Y*.MAC
 Routine(s): Y*.INT
 Routine(s): <RETURN>

 Long or Short form (L or S)? S=> <RETURN>
 Find routines last modified since date: 1/21/95
          and on or before date: t
 Display on
 Device: <RETURN>          Right margin: 80=> <RETURN>

      Short Listing of Selected Routine/Include Files
        Modified Between 1 July 95 and 10 July 95
                  Directory: /usr/msql
               10 Jul 95  01:24PM   Page 1

YAAA              --  .MAC  --

YAAA             --  .MAC.2  --

YAAA             --  .MAC.3  --

YBBB              --  .MAC  --

YDDD              --  .MAC  --

YAAA              --  .INT  --

YCCC              --  .INT  --
 >
```

When the Short Form display extends across multiple screens, %urdir pauses on each screen and displays the message "--more--" at the bottom of the screen. You may press <RETURN> to move ahead to the next screen.

Example 3

In the example below, %urdir uses the Long Form to display of all versions of the macro source, intermediate code, and object code routines and include files called “abc” located in the current directory:

```
>do ^%urdir

Routine(s): abc.*
Routine(s): <RETURN>

Long or Short form (L or S)? S=> L
Find routines last modified since date: <RETURN>
          and on or before date: <RETURN>
Display on
Device: <RETURN>         Right Margin: 80=> <RETURN>

    Long Listing of Selected Routines/include Files
             Directory /user/documentation
            10 Jul 95  01:25PM   Page 1

NAME   .EXT.VER  DATE/TIME             #BYTES    BLOCKS
abc    .MAC.1    10 Apr 91 10:35AM     1234
abc    .MAC.2    15 May 91 02:30PM     1255
abc    .MAC.3    25 Jun 91 08:45AM     1196
abc    .INT      25 Jun 91 08:48AM     1268
abc    .OBJ      25 Jun 91 09:00AM     1301      706

Total for Directory: /user/documentation
     -- .MAC --       3               3685
     -- .INT --       1               1268
     -- .OBJ --       1               1301      706
>
```

When the Long Form display extends across multiple screens, %urdir pauses on each screen and displays the message “--more--” at the bottom of the screen. You may press <RETURN> to move ahead to the next screen.

%urchange

The %urchange utility changes all occurrences of a specified string or strings in macro source routines, intermediate code routines, and include files to a new value. The new value may be the null string.

Note On ISM systems the %urchange utility is also known as %RCHANGE.

When you invoke %urchange, the utility prompts you to specify a search string and a replace string. You may specify multiple search strings with a corresponding replace string for each.

Next, the utility prompts you to specify the routines you want to change.

Note %urchange does not support remote directory syntax. This means that you cannot make changes to routines that reside outside your current directory.

In addition to searching for and replacing strings, the %urchange utility can also perform several auxiliary functions on modified routines, including:

- Generate a backup copy for each changed routine
- Recompile each changed routine
- Display a confirmation prompt for each change before it is made

Answer *Yes* at the appropriate prompt to enable any of these options. By default, %urchange recompiles all changed routines but does not generate backups or ask you to confirm each change.

If you answer *Yes* to the "Verify Each Change?" prompt, %urchange displays an "Okay to change?" confirmation prompt each time it reaches a line that contains a string to be changed.

%urchange automatically stamps the date and time when it modifies intermediate code routines.

Example

In the example below, %urchange changes every occurrence of the global "^ABD" to "^ABC" in all routines that begin with the letter "Z" and are located in the current directory.

```
>do ^%urchange

This routine changes all occurrences of a string in
routines/include files.

1.  Change every: ^ABD  to: ^ABC
2.  Change every:

Routine(s): Z*
Routine(s): <RETURN>

Generate backups? No=> <RETURN>
Recompile? Yes=> <RETURN>
Verify Each Change?  No=> <RETURN>

Display changes on
Device: <RETURN>            Right Margin: 80=> <RETURN>

              %urchang  Jan 31 95  2:42 PM

Changing "^ABD" to "^ABC"

ZRR.INC
ZRR+3   if ^ABC(0)=41250 set id=x1

ZAA.INT
ZAA+3   set ^ABC(0)=100

ZBB.INT
ZBB+5   set ^ABC(0)=41250

ZBB.MAC
ZBB+5   set ^ABC(0)=41250

>
```

The utility outputs the routine name, the text of the change, and the lines containing the changed string.

%urcomp

The %urcomp utility compiles macro source and intermediate code routines.

Note On ISM systems the %urcomp utility is also known as %RCOMPILE.

If you invoke %urcomp at the macro source level, it produces both intermediate and object code for the routine. Specifically, the macro preprocessor phase of the Compiler produces intermediate code, then the main Compiler produces the object code.

If you invoke %urcomp at the intermediate source code level, it directly produces object code.

When macro source routines include embedded SQL statements or make use of include files that contain embedded SQL statements, %urcomp comments out the SQL code. Thus, the intermediate code produced for a compiled macro source routine that contains embedded SQL statements consists of pure M source code with SQL statements as comments.

You should never compile routines that contain embedded SQL from the intermediate code level. If you attempt to do this, the Compiler issues an error message and aborts the compilation without modifying the routine. If a routine contains embedded SQL statements, you must compile it at the macro source level.

Note %urcomp does not support remote directory syntax. This means that you cannot compile routines that reside outside your current directory.

Example

In the example below, %urcomp compiles the macro source routine “EMPLIST”.

```
>do ^%urcomp

Routine(s): EMPLIST.MAC
Routine(s): <RETURN>

Display on
Device: <RETURN>            Right margin: 80=><RETURN>

EMPLIST.MAC
****** DECLARE deptcurs CURSOR FOR SELECT Department,Name
******    INTO :deptid,:deptname
******    FROM Department
*****     WHERE (%ALPHAUP (Name) %STARTSWITH :name)
****** DECLARE empcurs CURSOR FOR SELECT Name,Phone,
       HireDate
******    INTO :empname,:empphone,:empdate
******    FROM Employee WHERE (Department=:deptid)
****** SELECT Department,Name,COUNT(Department)
******    INTO :deptid,:deptname,:deptcount
******    FROM Department
******    WHERE %ALPHAUP(Name) %STARTSWITH :name
****** OPEN deptcurs
****** FETCH deptcurs
****** CLOSE deptcurs
****** OPEN empcurs
****** FETCH empcurs
****** CLOSE empcurs
****** CLOSE deptcurs
>
```

For the full text of the macro source routine “EMPLIST”, see Chapter 8, *Embedded SQL*.

%urcopy

The %urcopy utility lets you copy existing routines and include files.

Note On ISM systems the %urcopy utility is also known as %RCOPY.

You may use the %urcopy utility to do all of the following:

- Copy macro source routines, intermediate code routines, and include files within the current directory to other routine names and other extension names;
- Copy macro source routines, intermediate code routines, and include files from a foreign directory into the current directory;

  Note: Since the routine management utilities do not permit you to alter foreign directories, %urcopy does not allow you to copy routines from the current directory into a foreign directory.

- Compile macro source and intermediate code routines in the current directory;
- Generate backup versions of macro source routines and include files;
- Copy intermediate code routines to the macro source level in order to develop macro source code from routines that were created as pure M routines. This feature is particularly useful for users who are converting to Open M/SQL from earlier M versions;

When you invoke the %urcopy utility, you see the following prompt:

```
Copy routine(s):
```

Here you specify the name of the routine or include file you want to copy.

Note %urcopy supports remote directory syntax, which enables you to copy a routine from a directory other than the current directory. To do this, type the name of the source directory inside brackets and within quotation marks [“”], for example:

```
Copy routine(s): ["/us/mgr/tasks"]ABC.MAC
```

If you wish to again reference the same outside directory, you may simply type the carat character enclosed in brackets [^] at the next appearance of the “Copy Routine(s)” prompt. This recalls the name of the last explicitly referenced directory.

After you specify the routine/include file you want to copy, press <RETURN>. You see the following prompt:

```
To:
```

Here you specify the name to be given to the copied routine/include file. You may give the routine/include file any name (a new name or a name that already exists in the current directory) and any of the following extensions:

- .MAC
- .INT
- .INC

You may copy as many routines/include files as you want.

After specifying the routines/include files to copy and the new names for the copied versions, %urcopy prompts you for the following auxiliary options:

- Generate backups for each copied routine
- Recompile each copied routine
- Display errors encountered during compilation
- Overwrite existing routines with identical names

Answer *Yes* at the appropriate prompt to enable any of these options. By default, %urcopy recompiles all copied routines, displays all compilation errors, and overwrites existing routines with identical names. It does not generate backups by default.

Finally, %urcopy asks you to specify the device to which the routine names and status information should be output as the routines are copied.

Note You can compile a routine using the %urcopy utility by copying it to itself with the compile option enabled.

Example 1

In the example below, %urcopy copies all macro source routines beginning with the letter “A” located in directory “/USER/MSQL” into the current directory.

```
>do ^%urcopy
Copy routine(s): ["/USER/MSQL"]A*.MAC     To: A*.MAC
Copy routine(s): <RETURN>

Generate backups ? No=> <RETURN>
Compile? Yes=> N
Display Errors? Yes=> <RETURN>
Overwrite Existing Routines? Yes=> <RETURN>

Display routine names on
Device: <RETURN>           Right margin: 80=> <RETURN>

        AAA.MAC      ->     AAA.MAC
        ABC.MAC      ->     ABC.MAC
        ACC.MAC      ->     ACC.MAC
        AFG.MAC      ->     AFG.MAC

>
```

In this example, %urcopy does not compile the copied routines and does not generates backups for them. No routines are overwritten because the copied routines did not previously exist in the current directory.

Example 2

In this example, %urcopy copies the intermediate code routine “PRQ” to the intermediate code routine called “PZZ” and compiles the copied routine.

```
>do ^%urcopy
Copy routine(s): PRQ.INT      To: PZZ.INT
Copy routine(s): <RETURN>

Generate backups? No=> <RETURN>
Compile? Yes=> <RETURN>
Display Errors? Yes=> <RETURN>
Overwrite Existing Routines? Yes=> <RETURN>

Display routine names on
Device: <RETURN>          Right margin: 79=> <RETURN>
 PRQ.INT     ->    * PZZ.INT  —  COMPILED
>
```

This operation effectively replaces the previous contents of “PZZ” with the current contents of “PRQ”.

The asterisk shown in the output line of this example indicates that the copied routine has overwritten the previous contents of the target routine. The output line also displays the message “COMPILED” to indicate that the new routine has been compiled.

%urfind

The %urfind utility searches through macro source routines, intermediate code routines, and include files and returns all routine lines that contain at least one occurrence of any search string in a list of search strings. This contrasts with %urfand, which returns any routine lines that contain occurrences of **all** search strings in the list of search strings. %urfind does not operate at the object code level.

Note On ISM systems the %urfind utility is also known as %RFIND.

When you invoke %urfind, the utility prompts you to specify a search string. You may specify as many search strings as you want.

After specifying the search strings, %urfind displays the following prompt:

```
Exact Upper/Lowercase Match? Yes=>
```

Here you may answer *Yes* or *No* to indicate whether or not you want the search to be case-sensitive. *Yes* (case-sensitive) is the default response.

Next, the utility prompts you to specify the routines through which you want to search.

Example

In the example below, %urfind searches for occurrences of the strings “Hello” and “Goodbye” in all macro source and intermediate code routines located in the current directory whose names begin with the letter “A”.

```
>do ^%urfind
Find routine lines that contain at least one of a set of
strings.
1. Search for: Hello
2. Search for: Goodbye
3. Search for: <RETURN>

Exact Upper/Lowercase Match? Yes=> N

Routine(s): A*.MAC
Routine(s): A*.INT
Routine(s): <RETURN>

Display results on
Device: <RETURN>            Right Margin: 80=> <RETURN>

ABC.INT
ABC+4   S ^ABC(x) = "Hello"

AQQ.INTAYY.INT              AZZ.INT
AZZ+6   S ^ABC(z) = "Goodbye"

ABC.MAC
ABC+4   S ^ABC(x) = "Hello"

AQQ.MACAYY.MAC              AZZ.MAC
AZZ+6   S ^ABC(z) = "Goodbye"

>
```

The utility outputs the names of all routines through which it searches.

When it finds one of the specified search strings, it outputs the tag numbers and lines in which the string is located.

%urfand

The %urfand utility searches through macro source routines, intermediate code routines, and include files and returns routine lines that contain occurrences of **all** search strings in a list of search strings. This contrasts with %urfind, which returns any routine lines that contain an occurrence of at least one search string in the list of search strings. %urfand does not operate at the object code level.

Note On ISM systems the %urfand utility is also known as %RFAND.

When you invoke %urfand, the utility prompts you to specify a search string. You may specify as many search strings as you want.

After specifying the search strings, %urfind displays the following prompt:

```
Exact Upper/Lowercase Match? Yes=>
```

Here you may answer *Yes* or *No* to indicate whether or not you want the search to be case-sensitive. *Yes* (case-sensitive) is the default response.

Next, the utility prompts you to specify the routines through which you want to search.

Example

In the example below, %urfand searches for occurrences of the strings “Hello” and “Goodbye” in all macro source and intermediate code routines located in the current directory whose names begin with the letter “A”.

```
>do ^%urfand
Find routine lines that contain all of a set of strings.
1. Search for: ^ABC
2. And Search for: Hello
3. And Search for: <RETURN>

Exact Upper/Lowercase match? Y=> N

Routine(s): A*.MAC
Routine(s): A*.INT
Routine(s): <RETURN>

Display results
Device: <RETURN>            Right Margin: 80=> <RETURN>

ABC.INT
ABC+4   S ^ABC(z) = "Hello"

AQQ.INT     AYY.INT       AZZ.INT       ABC.MAC
ABC+4   S ^ABC(z) = "Hello"

AQQ.MAC     AYY.MAC       AZZ.MAC

>
```

The utility outputs the names of all routines that it searches.

When it finds all of the specified search strings in one line of a routine, it outputs the tag numbers and line in which the strings are located.

%urdel

You may use the %urdel utility to delete macro source routines, include files, intermediate code routines, and object code routines from the current directory and output a list of the deleted routines to a specified device.

Note On ISM systems the %urdel utility is also known as %RDELETE.

When you invoke %urdel, the utility prompts you to specify the routines you want to delete.

Note %urdel does not support remote directory syntax. This means you cannot delete routines/include files that reside outside your current directory.

When %urdel deletes one version of a routine, it shuffles the remaining versions forward. For example, suppose you maintain three versions of the macro source routine "ABC.MAC". When you delete version "ABC.MAC.1", the first backup version (previously called "ABC.MAC.2") is renamed to "ABC.MAC.1" and becomes the current version, and the second backup version is renamed to "ABC.MAC.2", while the second backup version ceases to exist.

Example

In the example below, %urdel deletes all versions of all routines (.MAC, .INT, .INC, .OBJ) located in the current directory that begin with the letter "Y".

```
>do ^%urdel
Delete routines/include files.
WARNING: When .MAC.1 is deleted, backups are shuffled
         forward, UNCOMPILED.

Routine(s): Y*.*.*
Routine(s): <RETURN>

Output on
Device: <RETURN>            Right margin: 80=> <RETURN>

         DELETE SELECTED ROUTINES/INCLUDE FILES

                   Jul 13 91  3:32 PM

                DIRECTORY: /DUA0/MARGARET

YAZ.MAC.1   YPP.MAC.1   YZZ.MAC.1   YAZ.MAC.2   YPP.MAC.2
YZZ.MAC.2   Y1.INC      Y2.INC      Y3.INC      YAZ.INT
YPP.INT     YZZ.INT     YAZ.OBJ     YPP.OBJ     YZZ.OBJ

>
```

%urverma

The %urverma utility lets you set the maximum number of backup versions maintained by the current directory for macro source routines and include files.

Note On ISM systems the %urverma utility is also known as %RVERMAX.

The default number of versions to maintain is four (one current version and three backups). The maximum number of versions that Open M/SQL can maintain is nine (one current version and eight backups).

Open M/SQL maintains backups for macro source routines and include files only. It does not maintain backups for intermediate code or object code routines; their version numbers are always .1.

Example

In the example below, %urverma is used to change the number of backup versions maintained for macro source routines and include files from the default value of 4 to the maximum value of 9.

```
>do ^%urverma

Number of versions to keep for .MAC: 4=> 9
Number of version to keep for .INC: 4=> 9

>
```

%urpurge

The %urpurge utility deletes backup versions of macro source routines and include files and outputs a list of purged routines and their version numbers to a specified device.

Note On ISM systems the %urpurge utility is also known as %RPURGE.

When you invoke %urpurge, the utility prompts you to specify the number of backup versions to be maintained after the purge. The default number of versions to be maintained is 1, the current version only. If you wish to keep any backup versions, you must change the default.

Next, the utility prompts you to specify the routines whose backups you want to purge.

Example

In the example below, %urpurge purges all backup versions of the macro source routines "TEST" and "MATT" in the current directory.

```
>do ^%urpurge

Purge backups, keeping how many versions: 1=> <RETURN>

Routine(s): TEST.MAC
Routine(s): MATT*.MAC

Device: <RETURN>           Right margin: 80=> <RETURN>

         PURGE SELECTED ROUTINES/INCLUDE FILES

                  RETAINING 1 VERSION
                  Jul 15 91  12:22 PM

                 DIRECTORY: /USER/MSQL

TEST.MAC.2   TEST.MAC.3   MATT1.INC.2  MATT1.INC.3

MATT1.INC.4  MATT2.INC.2  MATT2.INC.3  MATT3.INC.3

8 routines Purged.

>
```

%urpurge outputs a list of all routines and include files that it purges.

%urset

The %urset utility is used by the other Open M/SQL routine utilities to select routines.

You will likely never call this utility directly from the Open M/SQL programmer mode prompt. You may, however, find it useful to make calls to %urset in order to select routines from within a program.

Note On ISM systems the %urset utility is also known as %RSETN.

The Global ^mtemp

Routines selected using %urset are stored in the ^mtemp global.

The ^mtemp global has the following structure:

```
^mtemp(%msub,system@directory,extension,version,name)=""
```

The first subscript of the ^mtemp global is %msub, which is the variable returned by %urset.

The table below lists and describes the additional subscripts to the ^mtemp global:

Table 7-3: Subscripts of the ^mtemp Global

Subscript	Meaning
system@directory	System and directory where the routine resides.
extension	Extension of the routine (MAC, INC, INT, or OBJ).
version	Version number of the routine.
name	Name of the routine.

Parameters of %urset

The %urset utility has the following syntax:

```
^%urset(prompt,access,extensions,sort)
```

The following table lists and describes the four parameters accepted by %urset:

Table 7-4: Parameters of %urset

Parameter	Meaning
prompt	This parameter specifies the prompt used to ask for routines. The default prompt is “Routine(s):”. If this parameter is null, %urset uses the default prompt. **Note:** If your program uses a two-column format (like the format used by the %urcopy utility, with the “Copy From:” prompt in one column and the “Copy To:” prompt in a second column), you may use the delimiter $C(1) to separate the first and second pieces of the prompt.
access	This parameter specifies the type of access that is permitted for selecting routines. It may have the following three values: ▫ “D” — permits selecting of routines across directories on the same system. ▫ “S” — permits selecting of routines across directories and systems. ▫ Null — permits selecting of routines in the current directory only.
extensions	This parameter specifies the routine extensions that are accepted as valid. You may enter a list of the routine extensions, separated by commas. If this parameter is null, only routines with the extensions .MAC and .INT are accepted as valid.
sort	This parameter specifies the order in which selected routines are sorted. It may have either of the following two values: ▫ **DEVN** — sorts routines according to Directory/System, Extension, Version, and Name. ▫ **DNEV** — sorts routines according to Directory/System, Name, Extension, and Version. DEVN is the default sort order.

Example

In the example below, %urset is called with the following parameters:

- The prompt to be displayed for selecting routines is defined as:

  ```
  Routine Names:
  ```

- A null value is passed to the access parameter, meaning that access to routines across directories and systems is prohibited
- A value of “INT” is passed to the extension parameter to specify that .INT is the only extension accepted as valid for selecting routines
- A value of “DEVN” is passed to the sort order parameter to specify that the selected routines will be sorted by name

```
>do ^%urset(“Routine Names: ”,“”,“INT”,“DEVN”)

Routine Names: AAA.INT
Routine Names: BBB.INT
Routine Names: CCC.INT

>D ^%G

Global ^mtemp(%msub)

^mtemp(163,"@","INT",1,"AAA")=
        "BBB")=
        "CCC")=

>
```

PART

III

SQL Language Implementation

Chapter 8
Embedded SQL

Chapter 9
Open M/QSL Implementation of SQL

Chapter 10
Generating Queries

CHAPTER

8

Embedded SQL

Open M/SQL supports the embedding of SQL statements in macro source code.

This chapter describes the rules and syntax for using embedded SQL in Open M/SQL applications.

Specifically, it covers the following topics:

- Preprocessor Syntax Delimits Embedded SQL
- Open M/SQL Supports Two Kinds of Embedded SQL
- Referencing Macros in Embedded SQL
- Internal and External Values
- Multi-line Values
- Reserved Tag and Variable Names
- Portability
- Detailed Example

Preprocessor Syntax Delimits Embedded SQL

To embed SQL in Open M/SQL macro source code, you must delimit each SQL statement using the Open M/SQL preprocessor function &sql(...) or, alternatively, ##sql(...), for example:

```
&sql(DELETE *
      FROM Employees
      WHERE TerminationDate < 1/1/80)
```

The letters “sql” may be upper or lower case.

Open M/SQL Supports Two Kinds of Embedded SQL

Open M/SQL supports the following two kinds of embedded SQL statements:

- Non-cursor-based SQL
- Cursor-based SQL

Non-cursor-based SQL consists of individual SELECT, INSERT, UPDATE, and DELETE statements.

Cursor-based SQL is used for operations in which a program retrieves multiple rows from a table. To do this, the program declares and opens a cursor on the table, specifying one or more conditions on the rows to be retrieved. It then performs a series of fetches on the cursor, retrieving one row each time, until all the matching rows have been read.

Non-Cursor-Based SQL

A non-cursor-based embedded SQL SELECT statement query always returns a single row of data. Non-cursor-based SELECT statement queries are appropriate when you know that a single row of data matches the WHERE clause. If a non-cursor-based SELECT statement query matches more than one row, the query retrieves only the first of the matching rows and none of the rest, although it computes any aggregate functions in the SELECT list over all the matching rows.

Use the INTO Clause to Pass Retrieved Values to M Variables

A non-cursor-based SELECT statement query embedded in an M program always communicates its retrieved values to the program by using an INTO clause to read the values into M variables. For example:

```
&sql(SELECT Name,Telephone
        INTO :name,:tel
        FROM Employees WHERE SocSec=:ssn)
```

This example returns information about the employee whose social security number equals the value of the M variable “ssn”. If there is no such row, the query sets the local M variable SQLCODE to 100. If there is a matching row, the values of “Name” and “Telephone” are copied into the M variables “name” and “tel”. The names “name”, “tel”, and “ssn” in the above query are prefixed by colons, which indicates to the preprocessor that they are M variables.

The following example shows an embedded INSERT statement that adds a single row to the “Employee” table:

```
&sql(INSERT INTO Employee (Name,SocSec,Telephone)
        VALUES(“Boswell”,333448888,“546-7989”))
```

UPDATE and DELETE Statements Can Operate on Multiple Rows

Non-cursor-based UPDATE and DELETE statement queries can operate on multiple rows, for example:

```
&sql(UPDATE Employee SET AgeStatus=“MINOR”
        WHERE Age<21)
```

which sets “AgeStatus” to “MINOR” for every “Employee” row whose “Age” is less than 21, and:

```
&sql(DELETE FROM Products
        WHERE Supplier = :sup)
```

which deletes all “Products” rows whose “Supplier” field equals the variable “sup”.

For information on how to insert, update, and retrieve rows using arrays, see Chapter 9, *Open M/SQL Implementation of SQL*.

Cursor-Based SQL

When your application needs to access multiple rows of data, you must use a cursor. A cursor acts like a pointer—it focuses on accessing and processing one row at a time, then moves from that row to the next in the sequence.

Declaring a Cursor

Cursor-based SQL involves declaring one or more cursors on one or more base tables or views.

Note You may declare two or more cursors on the same base table and position them independently.

A cursor name may consist of any number of alphanumeric characters, though only the first six are significant. The first character must be alphabetic. All cursor operations for a given cursor name must reside in a single M routine.

To declare a cursor, you use the SQL DECLARE statement. Since DECLARE statements are not executable, InterSystems recommends for the sake of clarity that you place them at the beginning of a routine.

Once it has been declared, a cursor can be used by one or more OPEN, FETCH, and CLOSE statements. The DECLARE statement for a cursor must precede any other statements involving that cursor.

Below is an example of a DECLARE statement:

```
&sql(DECLARE PatCur CURSOR FOR
      SELECT Name,Telephone FROM Patients
      WHERE Name="Boswell")
```

This example declares a cursor "PatCur" that retrieves the names and telephone numbers of all rows in the "Patients" table with the name "Boswell". The DECLARE statement does not itself execute any code.

Opening a Cursor

Before using a cursor in a program, you must open it, as follows:

```
&sql(OPEN PatCur)
```

Depending on the particular query, the amount of code executed by a cursor OPEN can be very small or very large. When you finish using a cursor in a program, you should close it, as follows:

```
&sql(CLOSE PatCur)
```

Use FETCH to Retrieve Information into a Cursor

You use FETCH statements to retrieve information into a cursor. For example, the statement:

```
&sql(FETCH PatCur INTO :name, :tel)
```

fetches the next row as selected by the DECLARE statement for "PatCur" into the variables "name" and "tel". Note that the SELECT statement has already been specified as part of the DECLARE statement.

Use the INTO Clause to Pass Retrieved Values to M Variables

As in the non-cursor-based version, you must use an INTO clause to get values from the query into the program. In cursor-based SQL, you may specify the INTO clause as part of the DECLARE statement for a cursor, for example:

```
&sql(DECLARE PatCur CURSOR FOR
      SELECT Name, Telephone INTO :name, :tel
      FROM Patients WHERE Name="Boswell")
...
&sql(FETCH PatCur)
```

In this example, the FETCH statement retrieves values into the variables "name" and "tel".

Or, you may specify the INTO clause as part of each FETCH statement that uses the cursor, for example:

```
&sql(DECLARE PatCur CURSOR FOR
        SELECT Name, Telephone
        FROM Patients WHERE Name="Boswell")
...
&sql(FETCH PatCur INTO :name, :tel)
```

The ability to use the INTO clause in a FETCH statement is provided by ANSI-Standard SQL. The ability to use the INTO clause in a DECLARE statement is an InterSystems extension. For more information on INTO clauses, see Chapter 9, *Open M/SQL Implementation of SQL*.

You may also use an INTO clause in a FETCH statement to override the INTO clause in the corresponding DECLARE statement, for example:

```
&sql(DECLARE PatCur CURSOR FOR
         SELECT Name, Telephone INTO :name, :tel
         FROM Patients WHERE Name="Boswell")
...
&sql(FETCH PatCur INTO :n1, :t1)
```

Warning Do not specify %val as the host variable in the INTO clause for an SQL FETCH statement—this causes the FETCH statement to produce empty values. You may specify other "%" variables.

INSERT, UPDATE, and DELETE Operations Follow ANSI-Standard

Cursor-based INSERT, UPDATE, and DELETE operations follow ANSI-Stanadard SQL, for example:

```
&sql(DELETE FROM Patients WHERE CURRENT OF PatCur)
```

which deletes the row that was last FETCH'd using the cursor "PatCur".

Referencing Macros in Embedded SQL

Embedded SQL text may contain macro references. The Open M/SQL macro preprocessor expands these macro references before it translates the SQL text. Therefore, you may use macros to generate parts of the SQL text. The following example shows a series of macro definitions followed by an SQL statement that references the macros:

```
#define TABLE Patients
#define FIELDS Name,Phone
#define VARS :n,:p
#define COND Name %STARTSWITH "JO"
...
&sql(SELECT $$$FIELDS INTO $$$VARS
FROM $$$TABLE WHERE $$$COND)
```

Conversely, you may use a macro reference to insert the &sql preprocessor function, as in the following example:

```
#define GETNEXT &sql(FETCH xcurs INTO :a,:y)
   ...
   for i=1:1 $$$GETNEXT quit:SQLCODE=100  do ^process
```

When expanding the last line, the macro preprocessor first replaces the macro reference "$$$GETNEXT" with "&sql(FETCH...)" and then expands the embedded SQL expression.

Internal and External Values

Every value returned by an embedded SELECT query consists of the internal value (the value as stored in the database) plus an optional external value (if the Data Dictionary specifies internal-to-external conversion for the specific field).

If a value has an external value, it is separated from the internal value by the delimiter character $c(1). Therefore, every value returned by an embedded SQL query has one of two formats:

```
internal
```

or

```
internal_$c(1)_external
```

For a complete description of internal and external field values, see Chapter 9, *Open M/SQL Implementation of SQL*.

Multi-Line Values

When an INTO clause retrieves a multi-line (repeating) field into a variable, it puts the count of lines in the variable node and the value of each line in a numbered node under the variable.

For example, a two-line field value is stored as follows in the variable x:

```
x=2
x(1)=first line
x(2)=second line
```

For a complete description of multi-line fields, see Chapter 9, *Open M/SQL Implementation of SQL*.

Reserved Tag and Variable Names

When Open M/SQL generates intermediate code for a macro source routine that contains embedded SQL, it automatically generates tag and variable names.

For non-cursor-based SQL statements, all of the generated tag names begin with %0, and all of the generated variable names begin with %mmmsql.

For cursor-based SQL statements (suppose that operations are performed on a cursor called “PatCur”), all of the generated tag names begin with either %PatCur or %0, and all of the generated variable names begin with either %PatCur or %mmmsql.

Therefore, to avoid conflict with generated tag and variable names, macro source routines containing embedded SQL should not begin any tag names with %0 or any variable names with %mmmsql. They should also not begin tag or variable names with %PatCur, where PatCur is any cursor name used in the program.

Portability

If a macro source routine contains embedded SQL statements, you must compile the routine in the directory that contains the Data Dictionary definitions of all base tables referenced by the SQL statements.

After compiling a macro source routine that contains embedded SQL SELECT statements, you may copy the intermediate code and object code versions of the routine to another directory or system and run there without copying any other routines or globals, assuming the directory to which you copy uses a compatible database structure.

Note You cannot copy the intermediate code and object code versions of a macro source routine that contains any non-SELECT statement embedded SQL queries, i.e., you cannot copy routines that contain INSERT, UPDATE, or DELETE statements.

Detailed Example

The macro source routine shown below lists the employees in a company department. It asks the user to specify the first few characters of the department name. If there are no department matches with the characters provided by the user, the routine prints an error message and asks again. If there is one match, it lists the employees for that department. If there is more than one match, it uses cursor-based SELECT/FETCH operations to cycle through the department matches, asking the user which match is desired.

```
EMPLIST ;List the employees in a given department
        ;
         ;Define macro “CAPITAL” to convert a string to uppercase and remove
        ;punctuation. Calls the entry point alphaup^yraz as a
        ;function:
        ;
#define CAPITALS(%a) $$alphaup^yraz(%a)
        ;
        ;Define macro “EXTERNAL” to get the external value of a returned
        ;field:
        ;
#define EXTERNAL(%a) $piece(%a,$c(1),2)
        ;
        ;Declare cursor for accessing Department table. Selects fields
         ;”Department” and “Name” from Department table and puts them into
        ;M variables deptid and deptname:
        ;
        &sql(DECLARE dcurs CURSOR FOR SELECT Department,Name
                       INTO :deptid,:deptname
                       FROM Department
                       WHERE (%ALPHAUP(Name) %STARTSWITH :name))
        ;
        ;Declare cursor for accessing Employee table. Selects fields “Name”
        ;,”Phone”, and “Hiredate” into M variables empname, empphone,
        ;and empdate:
        ;
        &sql(DECLARE ecurs CURSOR FOR SELECT Name,Phone,HireDate
                       INTO :empname,:empphone,:empdate
                       FROM Employee WHERE (Department=:deptid))
        ;
        ;For formatting:
        ;
        kill set spaces=”                                                “
        ;
        ;Prompt user to select a department:
        ;
askd    write !,”Department name starts with: “ read name
        quit:name=””
        set name=$$$CAPITALS(name); to compare with
                                      %ALPHAUP(Name) below
        ;
        ;Non-cursor-based SELECT to get COUNT of Department rows whose
        ;name field begins with the value of the M variable name.
        ;COUNT goes into M variable deptcount, to be used below.
        ;Get the field values, too, in case COUNT is one:
        ;
        &sql(SELECT Department,Name,COUNT(Department)
                     INTO :deptid,:deptname,:deptct
                     FROM Department
                     WHERE %ALPHAUP(Name) %STARTSWITH :name)
         go:+deptct=0 nomatch  ;say “+deptct” because it might be null
        ;
```

```
         ;--------If there is only one, go to gotd and list the employees:
         ;
         go:deptct=1 gotd
         ;
         ;--------Otherwise, loop through the matching
         ;--------departments, asking the user which one to use:
         ;
         &sql(OPEN dcurs)

dloop    &sql(FETCH dcurs) if SQLCODE'=0 &sql(CLOSE dcurs) go nomatch
         write !,"Do you want the "_deptname_" department? "
         read answer go:answer="" dloop go:"Yy"'[$extract(answer) dloop
         &sql(CLOSE dcurs)
         ;
gotd     &sql(OPEN ecurs)
         write !,!,"Employees in the "_deptname_" department:"
         write !,"   NAME                    PHONE         HIRE DATE"
         write!,"_____________________________________________________"
         ;
eloop    &sql(FETCH ecurs) ;get values for the next employee in this dept
         if SQLCODE'=0 &sql(CLOSE ecurs) go askd
         write !,"  "_empname_$e(spaces,1,39-$1ength(empname))
         write empphone_$e(spaces,1,14-$length(empphone))
         write $$$EXTERNAL(empdate) ;External value of
                                    ;hire date (MM/DD/YY)
         go eloop
         ;----
nomatch  write "       No matching departments." go askd
```

CHAPTER

9

Open M/SQL Implementation of SQL

This chapter describes extensions to ANSI-Standard SQL that have been incorporated into Open M/SQL. It assumes that you are already familiar with standard SQL syntax and usage.

Specifically, it covers the following topics:

- Summary of Extensions
- Joins
- Aggregate Extensions
- Duplicate Rows
- Embedded SQL
- Using the INTO Clause
- Using the INTO Clause with Arrays
- VALUES Extension for INSERT and UPDATE Queries
- Multi-Line Fields
- Internal and External Values
- %INTERNAL and %EXTERNAL Functions
- M Operators
- Extensions to SQL Operators
- %STARTSWITH
- LIKE Predicate
- Collation Sequence
- %NOCHECK
- SQL Transaction Processing
- Privilege Operators
- Using Subqueries
- Query-Based Views

Summary of Extensions

Open M/SQL's implementation of Structured Query Language (SQL), referred to throughout this chapter and this guide as InterSystems' SQL, provides a variety of useful extensions to the ANSI-Standard SQL Data Manipulation Language (DML). These extensions enhance the power of SQL itself and help to integrate SQL with M.

The following list summarizes InterSystems' extensions to ANSI-Standard SQL:

- One-way outer join (=* operator)
- Implicit join (-> arrow syntax)
- Aggregate extensions (%FOREACH, %AFTERHAVING, DISTINCT BY)
- Embedded SQL
- Use of the INTO clause in cursor declaration
- SELECT INTO using mixed variables
- Array INSERT and UPDATE
- VALUES extension for INSERT and UPDATE queries
- Multi-line fields
- Internal and external values
- %INTERNAL and %EXTERNAL functions
- M operators
- LIKE predicate
- Starts-with operator (%STARTSWITH)
- Collation sequence functions (%ALPHAUP,%UPPER,%EXACT)
- %NOCHECK keyword
- SQL Transaction processing keywords (%BEGTRANS, %INTRANS)
- Privilege operators (GRANT, REVOKE, %CHECKPRIV)
- Subqueries
- Query-based view operators (CREATE VIEW, ALTER VIEW, DROP VIEW)

Added Keywords and Symbols

Keywords that represent InterSystems extensions to SQL begin with the percent sign (%). These keywords include:

- %AFTERHAVING
- %ALPHAUP
- %ALTER
- %BEGTRANS
- %CHECKPRIV
- %EXACT
- %EXTERNAL
- %FOREACH
- %FORM
- %INTERNAL
- %INTRANS
- %MENU
- %MEUNOBJ
- %NOCHECK
- %QUERY
- %REPORT
- %ROWCOUNT
- %STARTSWITH
- %THRESHOLD
- %UPPER

Note The keywords listed above are all Open M/SQL reserved words. To see a complete list of reserved words, refer to Appendix B of this guide.

The symbols shown in the following table also represent InterSystems extensions to SQL:

Table 9-1: SQL Extension Symbols

Symbol	Meaning
=*	One-way outer join
->	Implicit join
_ #	M scalar operators
? [	M comparison operators
& !	M AND and OR
]	M follows operator

InterSystems' SQL permits the use of the double quote characters ("...") to delimit literal values, in addition to the apostrophe character ('...') used by ANSI-Standard SQL.

InterSystems' SQL allows you to negate the comparison and logical operators by prefixing them with "not" or "NOT", for example:

```
not=  (equivalent to < >)

not<  (equivalent to >=)

not>  (equivalent to <=)

not?

not&

not[

not]

not!
```

Note The syntax "!=" is equivalent to "NOT=".

Joins

Joins provide the means of linking data in one table with data in another table and are frequently used in defining reports and queries.

A join is an operation that combines two tables to produce a third, subject to a restrictive condition. Every row of the new table must satisfy the restrictive condition.

One-Way Outer Joins

With standard “inner” joins, when rows of one table are linked with rows of a second table, a row in the first table that finds no corresponding row in the second table is excluded from the output table.

With one-way outer joins, all rows from the first table are included in the output table even if there is no match in the second table. With one-way outer joins, the first table pulls relevant information out of the second table but never sacrifices its own rows for lack of a match in the second table.

When specifying a one-way outer join, the order in which you name the tables in the FROM clause is very important. The first table you specify is the source table for the join.

You specify an outer join by using the symbol =* in place of = in the WHERE clause of the SQL query.

As an illustration of the difference between a regular join and a one-way outer join, consider the query:

```
SELECT Patient.PName,Doctor.DName
FROM Patient,Doctor
WHERE Patient.Doctor = Doctor.Doctor
```

This query returns one row for each row of the “Patient” table that has a non-null “Doctor” field. By contrast, examine the query:

```
SELECT Patient.PName,Doctor.DName
FROM Patient,Doctor
WHERE Patient.Doctor =* Doctor.Doctor
```

Though it differs only in the =* operator, this query returns all the rows of the previous query plus the rows of the “Patient” table that have null value for the “Patient.Doctor” field. For such rows, “Doctor.Name” is returned as null. This is known as “null padding”—Open M/SQL automatically performs null padding.

The one-way outer join condition expressed by the following syntax:

```
A.x =* B.y
```

specifies that every row in A be returned. For each A row returned, if there is a B row such that A.x=B.y, all of the corresponding B values are also returned. If there is no B row such that A.x=B.y, null padding causes all B values for that A row to return as null.

One-way outer join conditions, including the necessary null padding, are applied before other conditions. Therefore, a condition in the WHERE clause that cannot be satisfied by a null-padded value (for example, a range or equality condition on a field in B) effectively converts the one-way outer join of A and B into a regular join.

For example, if you add the clause “AND Doctor.Age < 45” to the two “Patient” table queries above, it makes them equivalent. However, if you add the clause “AND Doctor.Age < 45” or “Doctor.Age is null”, it preserves the difference between the two queries.

Implicit Joins

Implicit joins are pre-defined joins between tables which you specify in the Data Dictionary. They allow you to define queries without specifying the WHERE condition that is used to join tables.

On an internal level, an implicit join causes an additional join condition to be implicitly added (ANDed) to the WHERE clause and the joined table to be implicitly added to the FROM clause of the SQL query.

By definition, the implicit join is a one-way outer join. However, conditions on fields of the join table may cause the join to act as a regular join (as explained above).

Open M/SQL supports two types of implicit joins, **designative references** and **characteristic relationships**. Characteristic relationships can be either **child-to-parent** references or **parent-to-child** references.

Note You may only define designative and characteristic relationships among tables that reside within a single database.

Arrow Syntax Specifies Implicit Joins

You use arrow syntax to specify implicit joins between tables. Arrow syntax appears as a dash followed by a greater-than symbol (->).

In Open M/SQL, you may use arrow syntax in the following three cases:

1. To signify Designative References between tables
2. To signify Child-to-Parent References between tables
3. To signify Parent-to-Child References between tables

Note References to arrow syntax fields (such as a->b->c) may appear in the SELECT clause of a general view definition without the use of an AS clause. This holds true also for the SELECT clause of a subquery nested within the FROM clause of another query. For example, the following syntax is valid in Open M/SQL:

```
SELECT c FROM (SELECT a->b->c FROM d)
```

Designative References

A designative reference is a many-to-one link between tables in which one field of the designating table contains the Row IDs of all rows in the designated table. A designative reference is said to be a non-dependent link because rows in the referenced table exist independently of rows in the designating table. In relational database terminology, the designating table has a “foreign key” on the referenced table. In M terminology, the designating table has a “pointer” to the referenced table. In Open M/SQL, a field that designates another table is called a designative reference field.

The value of a designative reference field is the Row ID of a row in the designated table. Designative references provide a means of accessing information from a row of the designated table and using that information together with the associated row of the designating table.

If the field A.b is a designative reference field that designates table B and x is a field in B, the reference:

```
A.b->x
```

points to the value of x in the row of table B that corresponds to A.b. It is interpreted as a reference to B.x with B added implicitly to the FROM clause and an additional one-way outer join condition added implicitly to the WHERE clause, of the form:

```
A.b =* B.RowID
```

For example, the following query is equivalent to the second query on page 9-4:

```
SELECT Patient.PName,Patient.Doctor->DName
FROM Patient
```

The following query retrieves patient's name and patient's doctor's name for every patient who lives in Boston:

```
SELECT Patient.PName,Patient.Doctor->DName
FROM Patient
WHERE Patient.City = "Boston"
```

Assuming that "Patient.Doctor" is a designative reference to the "Doctor" table, this query is equivalent to:

```
SELECT Patient.PName,Doctor.DName
FROM Patient,Doctor
WHERE Patient.City = "Boston"
AND Patient.Doctor =* Doctor.Doctor
```

As another example, consider the query:

```
SELECT Patient.PName,Patient.Doc1->DName,Patient.Doc2->
      DName
FROM Patient
WHERE Patient.City = "Boston"
```

This query is equivalent to:

```
SELECT Patient.PName,DoctorX.DName,DoctorY.DName
FROM Patient,DoctorX,DoctorY
WHERE City = "Boston"
   AND Doc1 =* DoctorX.Doctor
   AND Doc2 =* DoctorY.Doctor
```

Note If it does not create ambiguity, A.b->x may be abbreviated as b->x.

Characteristic Relationships

In the Data Dictionary, you can create tables that are characteristic of other tables. This type of relationship is appropriate when one or more rows in the child table correspond to a single row in the parent table. For example, a parent table “Invoices” might have a child table “LineItems” containing one or more line items for each invoice. The child table contains a “parent reference” field, which is similar to a designative reference but points to the parent table.

A child table always designates its parent table. For this reason, a characteristic relationship can be thought of as a kind of designative reference. However, a characteristic relationship is more restrictive than a designative reference since the join condition specifies that all rows of the child table must designate the same parent table row. In this way, a characteristic relationship satisfies the relational definition of an inner join.

Child-to-Parent References

If table P is the parent of table C and x is a field in P, the implicit join syntax:

```
C.ParentRef->x
```

for a given C row refers to the value of x in that row's parent row. It is interpreted as a reference to P.x with P added implicitly to the FROM clause and an additional one-way outer join condition added implicitly to the WHERE clause, of the form:

```
C.ParentRef =* P.RowID
```

For example, given a parent table “Customer” with child table “Invoice”, the query:

```
SELECT Customer->Name
FROM Invoice
WHERE Number = 51140
```

is equivalent to:

```
SELECT Customer.Name
FROM Invoice,Customer
WHERE Invoice.Number = 51140
AND Invoice.Customer =* Customer.Customer
```

This query returns the name of the customer to whom invoice number 51140 belongs.

Parent-to-Child References

If P is the parent of C and x is a field in C, the implicit join syntax:

```
P.C->x
```

for a given row in P refers to the value of x in a child row of that row. It is interpreted as a reference to C.x with C added implicitly to the FROM clause and an additional one-way outer join condition added to the WHERE clause, of the form:

```
P.RowID =* C.ParentRef
```

Since there can be many children for a given parent, each child causes the query to produce another row.

For example, if you have a parent table “Customer” with child table “Invoice”, the query:

```
SELECT Invoice->Number
FROM Customer
WHERE Name = “Smith”
```

is equivalent to:

```
SELECT Invoice.Number
FROM Customer,Invoice
WHERE Customer.Name = “Smith”
AND Customer.Customer =* Invoice.Customer
```

This query returns the numbers of all invoices for all customers with the name “Smith”.

Aggregate Extensions

InterSystems provides extensions to the SQL aggregate functions AVG, COUNT, MAX, MIN, and SUM to make them more powerful and more flexible. These extensions simplify the relationship of the SQL aggregate functions with the GROUP BY and DISTINCT operators, effectively making aggregates, grouping, and duplicate elimination three orthogonal components of the SQL language.

Open M/SQL's aggregate function extensions are:

- Aggregates alongside fields in SELECT and HAVING clauses
- %FOREACH
- %AFTERHAVING
- DISTINCT BY in aggregates
- SELECT DISTINCT BY

Aggregates as Query Columns

The first extension allows aggregates to appear in a SELECT list or HAVING clause alongside field values. In ANSI-Standard SQL, if a query contains an aggregate, the SELECT and HAVING clauses cannot contain any field values outside of aggregates (unless they are listed in the GROUP BY clause). For example, the query:

```
SELECT Salary,AVG(Salary)
FROM Employee
```

is invalid in ANSI-Standard SQL because it selects both an aggregate and a field value.

InterSystems' SQL treats an aggregate simply as another column in each returned row, just as it treats a field or an expression composed of fields. In InterSystems' SQL, the above example returns one row of output for each row contained in the "Employee" table. The first column of each returned row is the salary of the corresponding employee. The second column is the value of the aggregate AVG(Salary), which is the same for each row and is the average salary for all employees. This feature is also useful in conjunction with the %FOREACH extension (see below).

The following query:

```
SELECT Name,Salary,AVG(Salary)
FROM Employee
HAVING Salary>AVG(Salary)
```

lists the name, salary and average salary for all employees whose salary is greater than the average salary. ANSI-Standard SQL does not accept such a query.

If the SELECT list consists only of aggregates and neither %FOREACH nor GROUP BY is used (in which case, the SELECT list is acceptable to ANSI-Standard SQL), Open M/SQL adds an implicit DISTINCT to the query so that it returns only one row. This is done because the value of the aggregate is the same for each selected row. For example, the query:

```
SELECT AVG(Salary)
FROM Employee
WHERE Department = "SALES"
```

returns the same value of AVG(Salary) for each "Employee" row, namely the average salary for all employees in the sales department.

Therefore, Open M/SQL adds an implicit DISTINCT to the query, as follows:

```
SELECT DISTINCT AVG(Salary)
FROM Employee
WHERE Department = "SALES"
```

This reduces the output to a single row consisting of the average salary in the sales department.

Two additional extensions generalize aggregates and make them more powerful. The %FOREACH keyword allows queries to produce different aggregate values for different groups of returned rows, and the DISTINCT BY feature provides a high degree of control over duplicate elimination.

%FOREACH

Obtaining aggregate results along with data is especially useful when combined with the InterSystems extension keyword %FOREACH, which can appear only inside an aggregate. The %FOREACH keyword divides the rows returned by a query into groups and calculates a separate aggregate value for each group.

%FOREACH uses the following syntax:

```
Aggregate (expr1 %FOREACH (expr2,expr3,...))
```

where a list of one or more arbitrary expressions follows the %FOREACH keyword.

For example, the query:

```
SELECT Name,Division,Department,
   AVG(Salary %FOREACH(Division,Department))
FROM Employee
```

returns four columns of information for each row in the "Employee" database.

The output columns for this query are shown in the table below:

Column	Value
1	Employee's name
2	Employee's division (call it DV)
3	Employee's department (call it DP)
4	Average salary for employees in division DV and department DP

ANSI-Standard SQL uses the keyword GROUP BY for a similar purpose. Whereas GROUP BY operates on an entire query, %FOREACH is more flexible because it only affects the value of a particular aggregate. Therefore, although Open M/SQL also supports ANSI-Standard SQL's implementation of GROUP BY, InterSystems recommends that you use %FOREACH and DISTINCT in all cases.

For another example of %FOREACH, the query:

```
SELECT PName,Ward,Age,AVG(Age %FOREACH(Ward)),Sex,
      AVG(Age %FOREACH(Sex))
FROM Patient
```

returns, for each patient, the average age of all patients in that patient's ward and the average age of all patients of that patient's sex:

The output columns for this query are shown in the table below:

PName	Ward	Age	AVG(Age) By Ward	Sex	AVG(Age) By Sex
Smith	6	59	62	F	59
Jones	5	16	37	M	29
Clark	5	42	37	M	29
Baker	6	65	62	F	59
Mulligatawny	5	53	37	F	59

%AFTERHAVING

The %AFTERHAVING keyword is another InterSystems extension that can appear inside aggregates within a SELECT clause.

%AFTERHAVING uses the following syntax:

```
Aggregate (... %AFTERHAVING)
```

When Open M/SQL encounters an aggregate with %AFTERHAVING, it computes the aggregate only for those rows that satisfy the query's HAVING clause.

For example, the query:

```
SELECT Name,Salary,AVG(Salary),AVG(Salary %AFTERHAVING)
FROM Employee
WHERE City = "Boston"
HAVING Salary > AVG(Salary)
```

computes two different salary averages. The aggregate "AVG(Salary)" computes the average salary for all Boston employees. The aggregate "AVG(Salary %AFTERHAVING)" computes the average salary for those Boston employees whose salary is greater than the average of all salaries for Boston employees.

Note Since InterSystems' SQL allows aggregates and fields to appear together in SELECT and HAVING clauses, you may compare the field "Salary" with the aggregate "AVG(Salary)" in the HAVING clause.

DISTINCT BY

InterSystems' SQL follows ANSI-Standard SQL by permitting the use of the keyword DISTINCT in a SELECT clause to remove duplicate rows from the output of a query. For example, the query:

```
SELECT DISTINCT Ward FROM Patient
```

returns one row for each distinct value of the field "Patient.Ward".

InterSystems' SQL also follows ANSI-Standard SQL by permitting the use of DISTINCT for aggregates in a SELECT clause, thus eliminating duplicate values from those used to compute the aggregate. For example, the query:

```
SELECT COUNT(DISTINCT City) FROM Patient
```

returns the number of distinct values in "Patient.City".

InterSystems' SQL provides an extension that allows you to eliminate just the rows that have duplicate values for specific expressions.

This extension is the use of DISTINCT followed by the keyword “BY” and a parenthesized list of expressions, specifying which fields and expressions should be distinct in the returned rows. Rather than eliminating rows that are duplicates in their entirety, it eliminates rows that have duplicate values for these expressions.

DISTINCT BY uses the following syntax:

```
DISTINCT BY (expr1, expr2, ...)
```

For example, the query:

```
SELECT DISTINCT BY(City) Mayor,Phone
FROM Precincts
```

returns the “Mayor” and “Phone” of one of the rows for each value of “City”. Note that the value of City itself is not selected by this query.

An example of DISTINCT BY with an aggregate function is:

```
SELECT SUM(DISTINCT BY(Name) Insurance)
FROM Employee
```

This query returns the sum of values for “Employee.Insurance”, including in the sum only one insurance value for each employee name.

The following is an example of DISTINCT BY using an expression:

```
SELECT DISTINCT BY(Salary + Commission) Salary,Commission
FROM Employee
```

This query returns one salary and commission for each value of salary + commission.

Duplicate Rows

ANSI-Standard SQL allows tables to contain multiple identical rows, i.e. rows that have the same values for all fields. InterSystems' SQL, however, does not permit identical rows. In accordance with most relational models, InterSystems' SQL requires that each table have a unique RowID field.

This extension eliminates potential problems with querying tables. Duplicate rows can cause otherwise well-defined queries and aggregates to return varying results depending on the order of the SQL clauses.

The functionality of duplicate rows is essentially preserved in Open M/SQL by a table design that allows rows to be identical except for a unique Row ID field that is automatically generated by the system (and may be custom-defined). This approach allows the creation of duplicate rows for otherwise unique fields such as social security numbers or company names, as they may be needed for look-ups, while still preserving the unique row concept of the relational model.

Embedded SQL

Open M/SQL allows the embedding of SQL statements in M programs. This feature replaces and enhances the Module Language (ML) of ANSI-Standard SQL.

You may embed SQL at the macro source code level only.

To embed SQL in Open M/SQL macro source code, you must delimit each SQL statement using the Open M/SQL preprocessor function &sql(...) or, alternatively, ##sql(...)

References to M variables within embedded SQL must begin with a colon.

Embedded SQL communicates retrieved information to M programs by using an INTO clause to read the values into M variables. For example, the embedded SQL query:

```
&sql(SELECT PName, Phone INTO :name,:tel
   FROM Patient
   WHERE SSN=:num)
```

uses the M variable “num” as input for the WHERE clause and the M variables “name” and “tel” to retrieve the output of the query.

For a complete description of how embedded SQL uses the INTO keyword to read query values into M variables, refer to Chapter 8, *Embedded SQL*.

An INTO clause may include both subscripted and unsubscripted M local variables as well as a mixture of both.

Embedded SQL statements can modify the M if-switch. Keep this in mind when embedding SQL inside an #if statement.

Using the INTO Clause

Both ANSI-Standard SQL and InterSystems' SQL permit the use of the INTO keyword in queries.

In non-cursor-based SQL queries, you may use the INTO clause with the SELECT statement, for example:

```
&sql(SELECT Name, Age INTO :n,:a
        FROM Employee WHERE SocSec = 555997777)
```

selects a row from the "Employee" table and puts the values of the "Name" and "Age" fields into the M variables "n" and "a".

Note Since this is a non-cursor-based embedded SQL query, it can retrieve only one row, even if more than one row matches the WHERE clause.

In cursor-based SQL queries, you may use the INTO clause with the FETCH statement, for example:

```
&sql(FETCH c INTO :x,:y,:z)
```

where "c" is a cursor name.

By using its own INTO clause each FETCH can read the same cursor into different sets of variables.

You May Use INTO in Cursor Declaration

An InterSystems' SQL extension permits the use of the INTO clause in the cursor declaration statement, for example:

```
&sql(DECLARE c CURSOR FOR
   SELECT Date, Time, Result INTO :d, :t, :r
   FROM Tests)
...
&sql(FETCH cursor)
```

Putting the INTO clause in the DECLARE statement (rather than the FETCH statement) produces slightly faster code and prevents having to repeat the INTO clause in each FETCH. However, you must be careful not to modify the variables referenced in the INTO clause while the cursor is open.

If you put an INTO clause in the DECLARE statement, you may override it with an INTO clause in a FETCH statement.

Using INTO with Arrays

InterSystems' SQL allows you to use the INTO clause with M arrays to do all of the following:

- Retrieve values from a SELECT query into an array
- INSERT values from an array into a table
- UPDATE a table with values stored in an array

SELECT INTO Using Arrays

InterSystems' SQL allows the INTO clause in a SELECT statement to retrieve values into an array.

This applies only to queries where all of the fields in the SELECT clause yield from a single table specified in the FROM clause.

To do this, the INTO clause must retrieve values into a subscripted M variable, and you must leave the last subscript of the M variable unspecified, for example:

```
INTO :a()
```

puts each retrieved field value into the array element:

```
a(col)
```

where "col" is the field's column number in the base table, as defined in the Data Dictionary. Similarly, the INTO clause:

```
INTO :a("foo",)
```

puts each retrieved field value into the array element:

```
a("foo",col)
```

For example, if the "Employee" table has the following fields:

Column	Name
1	Employees (RowID)
2	Name
3	Address (multi-line)
4	Department
5	Location
6	Telephone

the following embedded SQL query:

```
&sql(SELECT * INTO :emp()
         FROM Employee
         WHERE Name = "Smith")
```

reads the “Employees” field into emp(1), “Name” into emp(2), etc.

Since “Address” is a multi-line field, emp(3) is set to the number of address lines, emp(3,1) is set to the first line of “Address”, emp(3,2) to the second, and so on.

Caution Open M/SQL does not support use of the INTO clause to retrieve fields from a view into a local array. It only supports use of the INTO clause to retrieve fields from a base table into a local array.

SELECT INTO Using Mixed Variables

You can also mix subscripted and unsubscripted variables in the INTO clause of an SQL SELECT query. This allows you to use the array notation in a query involving expressions and/or fields from other tables. In this case, you must include additional scalar variables in the INTO clause to retrieve the values of expressions and fields from other tables.

When combining an array with other variables in an INTO clause, the array reference must always appear first. The array receives all values for fields from the table specified in the FROM clause. The remaining variables in the INTO list match up one-by-one with the expressions and fields from referenced tables in the SELECT clause. For example, the query:

```
SELECT PName, Doctor->DName, Sex, AVG(Age)
INTO :p(“items”,),:x,:y
FROM Patient
```

where “PName” is column 2 in “Patient” and “Sex” is column 5 in the “Patient” table, returns the following values:

```
p(“items”,2)=PName
p(“items”,5)=Sex
x=Doctor->DName
y=AVG(Age)
```

INSERT and UPDATE INTO Using Arrays

InterSystems' SQL allows you to use a variable reference with unspecified last subscript to pass an array of values into an embedded SQL INSERT or UPDATE query.

For example, the embedded SQL query:

```
&sql(INSERT INTO Employee
   VALUES :emp("profile",))
```

causes each field in the inserted "Employee" row to be set to:

```
emp("profile",col)
```

where "col" is the field's column number in the "Employee" table, as defined in the Data Dictionary.

Note that whereas ANSI-Standard SQL requires the table name to be followed by a list of field names, as in:

```
INSERT INTO Employee (Name,Telephone)
   VALUES (:name,:tel)
```

InterSystems' SQL does not require that you provide a list of field names after the table name when you use an array reference. The presence and absence of array elements determines the fields into which the query inserts values. This enables the contents of the array to dictate which fields receive insert values.

Using an Array Reference with an INSERT Query

ForINSERT queries, if an array entry is missing (undefined as opposed to null) and the corresponding field has an explicitly defined default value, the default value becomes the value of that field. However, if an array entry is explicitly defined as null, the default value does not override the null setting.

In the emp("profile") example above, if the "Employee" table has the following fields with corresponding default values as defined in the Data Dictionary:

Column	Name	Default
1	Employee	(Row ID)
2	Name	
3	Address	
4	Department	S
5	Location	
6	Telephone	

and the array values defined are:

```
emp(“profile”,2)=”Smith”
emp(“profile”,3)=2
emp(“profile”,3,1)=”1441 Main St.”
emp(“profile”,3,2)=”Cableton, IL  60433”
emp(“profile”,5)=””
emp(“profile”,7)=25
emp(“profile”,”next”)=”F”
```

the inserted “Employee” row has “Name” set to “Smith”, “Address” set to a two-line value, “Department” set to the default “S”, and “Location” set to null. The default value for “Location” is not used since the corresponding array element is defined with a null value. The array elements “7” and “next” do not correspond to column numbers in the “Employee” table, therefore the query ignores them.

Using an Array Reference with an UPDATE Query

An UPDATE query can also reference an array with unspecified last subscript. Whereas INSERT uses the presence and absence of array elements to assign values and default values to a newly created row, UPDATE uses the presence of an array element to indicate that the corresponding field should be updated. For example, consider the following embedded SQL statement:

```
&sql(UPDATE Employee
        VALUES :emp(“profile”,)
        WHERE Employee = 379)
```

Given the above definitions and array values, this statement will update the values of the “Name” and “Location” fields of the “Employee” row for which Row ID = 379.

You may also use an array reference with an UPDATE query that targets multiple rows, for example:

```
&sql(UPDATE Employee
        VALUES :emp(“profile”,)
        WHERE Type = “PART-TIME”)
```

As with INSERT, the contents of the array dictate which fields receive update values.

VALUES Extension for INSERT and UPDATE Queries

Typically, INSERT queries use the following syntax:

```
INSERT INTO <tablename> (<fieldname>,<fieldname>,...)
   VALUES (...,...)
```

and UPDATE queries use the following syntax:

```
UPDATE <tablename> (<fieldname>,<fieldname>,...)
   VALUES (...,...)
```

where the elements in the VALUES clause correspond in sequence to the fields specified after the table name.

Note If there is only one value element specified in the VALUES clause, it is not necessary to enclose the element in parentheses.

InterSystems' SQL allows INSERT and UPDATE queries to use either syntax above without requiring you to explicitly specify a list of field names after the table name.

In order to skip the step of specifying a list of field names after the table name, your query must meet the following two criteria:

1. The number of values specified in the VALUES clause is the same as the number of fields in the table.
2. The values in the VALUES clause are listed in order of the internal column numbers of the fields.

For example, the query:

```
INSERT INTO tabl1 VALUES (5,"John")
```

is equivalent to the query:

```
INSERT INTO tabl1 (age,name) VALUES (5,"John")
```

if the table "tabl1" has exactly two fields.

In this example, the value 5 is assigned to the field with the lower column number, and the value "John" is assigned to the other field.

Multi-Line Fields

Open M/SQL extends the relational database model by permitting the creation of multi-valued fields, commonly called multi-line fields by M programmers. Such fields are useful for storing information about a single entity where that information spans several lines. A typical multi-line field might be used for an address or a block of comment text. While such structures are useful, they violate the First Normal Form relational precept and, therefore, are not relationally correct. The user should be aware that SQL contains no operators for manipulating multi-line fields, and that it is impossible to perform comparisons or row ordering with multi-line fields.

InterSystems' SQL allows you to use multi-line fields in both output queries (SELECT) and input queries (INSERT and UPDATE).

Using Multi-Line Fields in SELECT Queries

In SELECT queries, you may reference multi-line fields in the SELECT statement by using an INTO clause to copy the contents of the field into a variable.

Note You may not use multi-line fields with comparison operators in the WHERE clause or in any of the row ordering clauses. If you attempt to use a multi-line field in a WHERE clause or in a row ordering clause, Open M/SQL returns an error message when it tries to process the query.

When an INTO clause copies a multi-line field into a variable, it creates an array by putting the count of lines in the variable node and the value of each line in a descendant node under the variable. For example, if the field "Patient.Address" for patient row 758 contains the following three lines:

```
426 Sunday Drive
Apt. 12C
Roberta, CA  90126
```

the embedded query:

```
&sql(SELECT Address INTO :a
        FROM Patient
        WHERE Patient = 758)
```

places the following values in "a":

```
a=3
a(1)="426 Sunday Drive"
a(2)="Apt. 12C"
a(3)="Roberta, CA  90126"
```

Similarly, the query:

```
&sql(SELECT Address INTO :b("addr")
        FROM Patient
        WHERE Patient = 758)
```

places the following values in "b":

```
b("addr")=3
b("addr",1)="426 Sunday Drive"
b("addr",2)="Apt. 12C"
b("addr",3)="Roberta, CA  90126"
```

Using Multi-Line Fields in INSERT and UPDATE Queries

You may also target multi-line fields with INSERT and UPDATE queries by using an array in the VALUES clause as input to the INSERT or UPDATE statement. For example, the query:

```
&sql(UPDATE Doctors(Address)
        VALUES :b("addr")
        WHERE Doctors = :doc)
```

which uses the "b" values listed above as input to the multi-line field "Doctors.Address".

Since "Doctors.Address" is a multi-line field in this example, the variable b("addr") must contain a multi-line value as above. If, however, the input value is not a variable (for example, if it is a constant), it is used as the first line of a multi-line value. For example, the query:

```
&sql(UPDATE Doctors(Address)
        VALUES "Main Street"
        WHERE Doctors = :doc)
```

sets "Doctors.Address" to a one-line value whose first line is "Main Street".

Internal and External Values

The concept of internal and external values pertains to fields that are defined in the Data Dictionary to have External-to-Internal and/or Internal-to-External Conversion Code. The “internal” value is the value that is stored in the database. The “external” value is a printable or user-readable form.

By default, the following Open M/SQL field types have conversion code and maintain both internal and external values:

- Number
- Date
- Time
- Yes/No

And, the following field type has the option to generate conversion code for maintaining both internal and external values.

- Multiple Choice

For example, the internal value for a field of data type Date has the M format of $H and the external value is a printed representation that depends on which of six date formats the application developer chooses for that field. Furthermore, you can explicitly specify additional conversion code by inserting M code.

In SQL queries, the following rules apply concerning internal and external values:

1. For a given field, the internal value is the first piece of $c(1) and the external value is the second piece of $C(1), as follows:

   ```
   Internal_$c(1)_External
   ```

2. Variables and constants are neither internal nor external and have no data types. Their use depends on context. When comparing a field with a variable, the variable should contain a value suitable for comparison with the field’s internal value.

3. As an exception to rule 2, constants are treated specially in conditions such as:

   ```
   Field <Op> Constant
   ```

   or

   ```
   Constant <Op> Field
   ```

   where <*Op*> is any comparison operator or where <*Op*> specifies input to a field in an INSERT or UPDATE statement.

In this context, the field's external-to-internal conversion code is applied to the constant. This allows a comparison such as:

```
StartDate > "06/01/86"
```

to test whether or not the internal value of the field "StartDate" is greater than the internal representation of "06/01/86", i.e. whether "StartDate" is after 06/01/86.

4. Variables are also treated specially in the same two conditions, namely:

   ```
   Field <Op> Variable
   ```

   or

   ```
   Variable <Op> Field
   ```

   where <*Op*> is any comparison operator or where <*Op*> specifies input to a field in an INSERT or UPDATE statement.

   In this context, the variable can optionally be given an internal and external value by using the delimiter character $c(1). If the variable's value is:

   ```
   Internal_$c(1)_External
   ```

   or simply:

   ```
   Internal
   ```

   where "Internal" has a non-null value, "Internal" is taken as the internal value for either comparison with a field or input to a field.

   If the value is:

   ```
   $c(1)_External
   ```

   the field's external-to-internal conversion is applied to "External" to produce an internal value.

5. For all fields that have internal-to-external conversion code defined in the Data Dictionary, each value returned by an embedded SELECT query also takes the form:

   ```
   Internal_$c(1)_External
   ```

   If there is no internal-to-external conversion code defined in the Data Dictionary, the value returned is simply

   ```
   Internal
   ```

   In either case, the internal value is:

   ```
   $piece(value,$c(1),1)
   ```

The following example uses both internal and external values returned by a query:

```
&sql(SELECT HireDate INTO :hiredate
   FROM Employee
   WHERE SocSec = 426713280)
set intdate=$piece(hiredate,$c(1))
set extdate=$piece(hiredate,$c(1),2)
if intdate>$h write !,"Not hired until",extdate
```

6. Each value returned by a non-embedded SELECT query is the result of applying any applicable internal-to-external conversion code.

%INTERNAL and %EXTERNAL Functions

InterSystems' SQL supports the following two SQL unary functions:

- %EXTERNAL
- %INTERNAL

When applied to a field for which Internal-to-External conversion code is defined, %EXTERNAL returns the external value for that field.

When applied to a field for which Internal-to-External conversion code is defined, %INTERNAL returns the internal value for that field.

You may reference the %EXTERNAL and %INTERNAL keywords in the SELECT, WHERE, and ORDER BY clauses of an SQL query, using the following syntax:

```
SELECT %EXTERNAL(FieldName)
```

Or, you may specify the field by its column number, as follows:

```
SELECT %EXTERNAL(FieldColumnNumber)
```

For example:

```
SELECT %INTERNAL(FieldName),...
FROM Table
WHERE %INTERNAL(FieldName) = 'some value' ...
ORDER BY %INTERNAL(FieldName)
```

%EXTERNAL enables you to retrieve and manipulate the external value of a field. For example, you might define conditions based on the external value of a field, or you might define a field to sort by its external value. This is especially useful for Multiple Choice fields, where the internal value is a number, but you need to test or sort by the external value.

%INTERNAL enables you to retrieve and manipulate the internal value of a field. For example, you might define conditions based on the internal value of a field, or you might define a field to sort by its internal value. This is especially useful for Multiple Choice fields, where the external value is a number, but you need to test or sort by the internal value.

If the field does not have a separate external value (i.e., Internal-to-External conversion code is not defined), %INTERNAL and %EXTERNAL both return the field's internal value. This makes it possible to use these keywords even when you do not know whether or not a field has Internal-to-External conversion code defined for it.

M Operators

InterSystems has extended SQL to support several M operators. This increases the expressive power of SQL and helps to better integrate the two languages.

The table below lists and describes the M operators supported by InterSystems' SQL:

Table 9-2: M Operators Supported By InterSystems' SQL

Operator	Usage
_ (space followed by underscore)	Concatenates two strings wherever it is not part of an identifier. You must use a space to prevent ambiguity. The expression: A_B is the name A_B containing an underscore. While the expression: A _B is the concatenation of A with B.
& !	For compatibility with the M language, InterSystems' SQL allows "&" as a synonym for AND and "!" as a synonym for OR.
? # / \ [ ]	InterSystems' SQL also recognizes the following M operators: ▪ ? — pattern match ▪ # — modulo (remainder) ▪ / — division ▪ \ — integer division ▪ [— string contains ▪] — follows
=	InterSystems' SQL implements the equals sign in accordance with the M convention rather than the ANSI-Standard SQL convention for comparing text strings of unequal length. Whereas ANSI-Standard SQL pads the shorter text string with blank spaces, InterSystems' SQL does not. Therefore, ("a " = "a") is false.

Table 9-2: M Operators Supported By InterSystems' SQL (Continued)

Operator	Usage
NOT or not	You may negate comparison and boolean operators by prefixing them with the word "not" or "NOT" (without intervening spaces). You may also use the word "not" as a prefix for an entire condition. The expressions: `WHERE NOT Age < 5` and `WHERE Age NOT < 5` are synonymous. **Note:** You cannot use the M apostrophe (') to signify NOT because SQL uses the apostrophe to signify quotation marks for literal strings.
!=	InterSystems' implementation of SQL supports the syntax "!=". This syntax is equivalent to "NOT=".

Pattern Match Operator Can Test Variables

InterSystems' SQL allows you to use the pattern match operator (?) against user-defined variables.

To do this, you must prefix the variable name with the indirection symbol "@".

For example, suppose you set the variable "namevar" to a pattern match condition, as follows:

```
S namevar="1.10A1","1.5A"
```

You could use the pattern match operator to reference this pattern match condition in the WHERE caluse of a query, as follows:

```
SELECT Name
   FROM Employees
   WHERE Name ? :@namevar
```

This query retrieves names from the "Employees" table for all rows where the "Name" field has the form:

```
<1-10 alphabetic characters>,<1-5 alphabetic characters>
```

such as:

```
Smith,John
```

but not:

```
Washington,George
```

Extensions to SQL Operators

Open M/SQL provides extensions to several SQL operators, as described below.

[NOT] IN Operator

InterSystems' SQL supports the use of the [NOT] IN operator with fields of data type Date and Time in WHERE clause conditions. The [NOT] IN operator tests whether or not a date or time falls within a specified date/time range.

For Date fields, the [NOT] IN operator uses the following syntax:

```
Date_Field [NOT] IN (Start_Date,End_Date)
```

where *Start_Date* is a start date in external format, and *End_Date* is an end date in external format.

For Time fields, the [NOT] IN operator uses the following syntax:

```
Time_Field [NOT] IN (Start_Time,End_Time)
```

where *Start_Time* is a start time in external format, and *End_Time* is an end time in external format.

For example:

```
SELECT *
   FROM Table
   WHERE Date_Field IN ('10/25/1994','01/26/93') AND
   Time_Field IN ('10:25PM','11:35AM')
```

Concatenation Operator

InterSystems' SQL supports the use of the SQL concatenation operator (||).

For example, if the fields "F1" and "F2" have the following values:

```
F1 = "Part One", F2 = "Part Two"
```

the query:

```
SELECT F1,F2,F1||' & '||F2 INTO :val1,:val2,:val3
   FROM Table
   WHERE F1||' & '||F2='Part One'||' & '||'Part Two'
```

yields the following results:

```
val1 = "Part One"
val2 = "Part Two"
val3 = "Part One & Part Two"
```

%STARTSWITH

InterSystems' SQL provides the extension keyword %STARTSWITH.

%STARTSWITH is a comparison operator that tests to determine whether one character string is a prefix of another.

%STARTSWITH uses the following syntax:

```
expr1 %STARTSWITH expr2
```

where the expressions can be any valid SQL expressions.

For example, a query that contains the WHERE clause:

```
WHERE ProductCode %STARTSWITH "S"
```

returns all rows where the value of the "ProductCode" field starts with the character "S".

%STARTSWITH treats fields of data type Name somewhat differently. With fields of data type Name, %STARTSWITH can match both the last name and first name strings on either side of the comma.

For example, if "EmpName" is a field of data type Name, then a query that contains the WHERE clause:

```
WHERE EmpName %STARTSWITH "SM,J"
```

returns all rows for which $PIECE(EmpName,"","",1) starts with "SM" and $PIECE(EmpName,"","",2) starts with "J". For example:

```
EmpName = "SMITH,JANE"
EmpName = "SMOOT,JERRY"
```

but not:

```
EmpName = "SMITH,HERMIONE"
```

The %STARTSWITH operator and other comparison operators are modified by collation sequences as described in the section entitled "Collation Sequence" on page 9-37.

LIKE Predicate

Open M/SQL supports the ANSI Standard SQL-2 LIKE predicate for performing comparison tests on two values.

The LIKE predicate uses the following syntax:

```
<match value> [NOT] LIKE <search pattern> [ESCAPE <escape character>]
```

Where the syntax elements have the following meanings:

1. The value for <*match value*> may be a field or a literal string. If it is a literal string, it must be enclosed in single quotes.
2. The value for <*search pattern*> can be either a literal string or a host variable. If it is a literal string, it must be enclosed in single quotes. Literal strings may reference the special pattern matching characters "%" and "_".
3. If the ESCAPE clause is specified, the value for <*escape character*> must be one and only one character. Also, if the escape character is present in the search pattern string, then the character following the escape character within the pattern string must be one of the special pattern matching characters, either "%" or "_".

Below is a sample SQL query that uses the LIKE predicate in the WHERE clause:

```
SELECT FieldA, FieldB, FieldC
FROM Table
WHERE FieldA LIKE 'abc%'
```

This query retrieves "FieldA", "FieldB", and "FieldC" from "Table" for all rows where "FieldA" starts with the character string "abc" and has any number of characters in length.

Note By default, the LIKE predicate always uses the external value of a field for comparison purposes.

Note The LIKE predicate uses the collation function (EXCAT, ALPHAUP, UPPER) used by the match value on the left side of the equation to perform the comparison test.

Special Pattern Matching Characters

The LIKE predicate supports the following two special pattern matching characters:

- "_" (underscore symbol)
- "%" (percent sign)

These special pattern matching characters are defined as follows:

"_" You may use this special character in the search pattern string to match any single character in the same position in the match value string.

Examples include:

```
'>Match This' LIKE '_Match This'          (Match)
'>>Match This' LIKE '_Match This'         (No Match)
'Match This' LIKE '_Match This'           (No Match)
```

"%" You may use this special character in the search pattern string to match any number of characters (zero or more) between the last exact matching character and the next exact matching character in the match value string.

Examples include:

```
'>Match This' LIKE '%Match This'          (Match)
'>>Match This' LIKE '%Match This'         (Match)
'Match This' LIKE '%Match This'           (Match)
'Match This!' LIKE '%Match This'          (No Match)
'Match??? This!!!' LIKE 'Match% This%'    (Match)
```

Specifying a Host Variable as the Search Pattern

The search pattern you specify after the LIKE predicate does not always have to be a string literal. You may also specify a search pattern that references a host variable, for example:

```
'Match This' LIKE :hostvar
```

ESCAPE Qualifier

Open M/SQL's implementation of the LIKE predicate supports the ESCAPE qualifier. The ESCAPE qualifier lets you suppress the meaning of the special characters "%" and "_" in the search pattern string and treat them instead as literal characters.

For example, the following SQL statement:

```
'Match This' LIKE '$_Match This' ESCAPE '$'
```

evaluates to the following M code:

```
"Match This"?1"_Match This"
```

which is not true.

However, the following SQL statement:

```
'Match_This' LIKE 'Match$_This' ESCAPE '$'
```

evaluates to the following M code:

```
"Match_This"?1"Match_This"
```

which is true.

Error Handling

If the syntax of the LIKE predicate contains a mistake, Open M/SQL returns the error code SQLCODE=-63, and you see the following error message:

```
Data Exception - invalid escape character
```

Collation Sequence

Collation sequence is the ordering of character strings.

Open M/SQL supports six collation sequence functions:

- EXACT
- ALPHAUP
- UPPER
- Plus (+)
- Minus (-)
- Space ( )

These collation sequence functions determine the ordering of the output values produced by SELECT queries as well as the behavior of comparison operations.

EXACT

The EXACT collation sequence orders pure numeric values (values for which x=+x) in numeric order first, followed by all other characters in string order. EXACT uses the same collation sequence for strings as the ANSI-Standard ASCII collation sequence. According to the ASCII collation sequence, digits are collated before uppercase alphabetic characters and uppercase alphabetic characters are collated before lowercase alphabetic characters. Punctuation characters occur at several places in the sequence.

Note Numbers that are not canonic (e.g., 02 or 1.30) collate as strings rather than numbers.

The following example shows several strings listed in EXACT collation sequence:

String
A'Ha
ARNOLD
Adams
a'Choo
aaronson

Note that "'" (apostrophe) collates before "A".

EXACT is the default collation sequence function. If you prefer another collation sequence, you may choose one of the collation sequences described below.

ALPHAUP

The ALPHAUP collation sequence function converts all strings to ALPHAUP format and then collates or compares according to the EXACT collation sequence.

ALPHAUP format removes all punctuation (non-alphanumeric) characters except commas and question marks and converts all alphabetic characters to uppercase.

The following example shows the same strings as above listed in ALPHAUP collation sequence:

String	Compared As
aaronson	AARONSON
a'Choo	ACHOO
Adams	ADAMS
A'Ha	AHA
ARNOLD	ARNOLD

UPPER

The UPPER collation sequence function converts all strings to UPPER format and then collates or compares according to the EXACT collation sequence.

UPPER format converts all alphabetic characters to uppercase but leaves punctuation characters intact.

The following example shows the same strings as above listed in UPPER collation sequence:

String	Compared As
a'Choo	A'CHOO
A'Ha	A'HA
aaronson	AARONSON
Adams	ADAMS
ARNOLD	ARNOLD

Plus, Minus, and Space

The Plus, Minus, and Space collation sequence functions are directed towards the collation of numeric values.

The table below describes the Plus, Minus, and Space collation sequence functions:

Table 9-3: Plus, Minus, and Space Collation Sequence Functions

Function	Effect On Collation Sequence	Index Map Syntax
Plus (+)	Deletes leading character zeros so that numeric values with leading character zeros collate as numbers rather than as character strings.	+{field}
Minus (-)	Reverses the collation sequence for numeric values. This causes numeric values to collate in descending order (highest to lowest) instead of ascending order (lowest to highest).	-{field}
Space ( )	Forces all values to collate as character strings. This causes all numeric values to collate after all character strings.	" "_{field}

Field Collation Sequence

In Open M/SQL, all fields of data type Name and Text use one of the six collation sequence functions described above (EXACT, ALPHAUP, UPPER, Plus, Minus, or Space).

Whenever you define a base table field in the Data Dictionary of data type Name or Text, you assign a collation sequence function to it. The default collation sequence for all Name and Text fields is EXACT.

The collation sequence function you assign to a field affects the results of SQL ORDER BY clauses involving that field. Choice of collation sequence also affects comparisons on that field using any of the comparison operators, listed below:

```
<
>
=
>=
<=
!=
not>
not<
%STARTSWITH
```

Collation Sequence and ORDER BY

When an SQL query has an ORDER BY clause that names one field with a data type of Text or Name, as in:

```
SELECT EmpName ...
ORDER BY EmpName
```

the query uses the collation sequence function defined for the order-by field to determine the order in which it returns matching rows.

If a query has an ORDER BY clause that names any other field or expression, as in:

```
SELECT DateEnd-DateStart ...
ORDER BY 1
```

the query collates its output according to the EXACT collation sequence function.

If the field “EmpName” uses the ALPHAUP collation sequence function, the example:

```
SELECT DateEnd-DateStart,EmpName ...
ORDER BY 1,2
```

orders the returned rows first by “DateEnd-DateStart” in EXACT collation order, and within that by “EmpName” in ALPHAUP order.

Collation Sequence and Comparisons

When performing comparisons of character string values, Open M/SQL typically uses the EXACT collation sequence function.

If either side of a comparison operator is a field of data type of Name or Text, Open M/SQL uses the collation sequence function defined for the field in the Data Dictionary. For example, if “EmpName” is defined to use the ALPHAUP collation sequence function, the comparison:

```
EmpName > “jo”
```

tests “EmpName” in ALPHAUP format to determine whether it is greater than (follows in collation sequence) the string “JO”. In effect, Open M/SQL implicitly applies the ALPHAUP function (see %ALPHAUP below) to both sides of the comparison to convert each side to ALPHAUP format. After the conversion, it performs the comparison based on the EXACT collation sequence.

If both sides of a comparison are fields of data type Name or Text, Open M/SQL uses precedence order to choose the collation sequence for the comparison. The precedence order is:

1. ALPHAUP (highest)
2. UPPER
3. EXACT (lowest)

Open M/SQL chooses the collation sequence with the higher precedence.

%ALPHAUP, %UPPER, and %EXACT

InterSystems' SQL supports three extension functions to force the conversion of a value to ALPHAUP, UPPER, or EXACT format, as described above.

These extension functions are:

- %ALPHAUP
- %UPPER
- %EXACT

Suppose that the field "EmpName" is defined to use the ALPHAUP collation sequence. The example:

```
%ALPHAUP(EmpName) > "jo"
```

compares "EmpName" converted to ALPHAUP format with the lowercase string "jo". Since %ALPHAUP(EmpName) is an expression rather than a field, Open M/SQL uses the EXACT collation sequence to perform the comparison.

Similarly, the example:

```
%EXACT(EmpName) > "jo"
```

compares the EXACT value of "EmpName" with the lowercase string "jo".

In other contexts, the %ALPHAUP and %UPPER functions simply convert individual values. The SELECT clause:

```
SELECT %ALPHAUP(EmpName)_" -- "_Phone
```

returns a value consisting of the "EmpName" field converted to ALPHAUP format concatenated with the constant " -- " concatenated with the EXACT value of the "Phone" field, for example:

```
RADCLIFF,CHRIS -- 723-8255
```

Changing the Default Collation Sequence

Under some circumstances, you may want to change the default M collation sequence to be something other than EXACT. Though it is possible, you should do this only with extreme caution. Three possible reasons for changing the EXACT collating sequence are:

1. To handle global data that was created previously with a particular collation sequence.
2. To build an application that uses a national character set different from the default character set.
3. To build an application requiring non-standard ordering of strings containing numbers.

Changing Collation Sequence on ISM Systems

Collation sequence consists of two components:

1. Character set — This is a string of ASCII character codes that gives the members of the set and their collation order.
2. String collation switch — If ON, the collation order applies to all strings including numbers. If OFF (default), canonic numeric strings (x=+x) are collated first in numeric order followed by all other strings in collation order.

You can change these two components independently, as follows:

- For a particular process, the $ZU(23) function changes the character set and string collation switch for the M variable] (“follows”) operator and for $ORDER on local variables.
- For a particular global, you may use the %GCREATE utility to create n M global with a revised character set and string collation switch. You cannot change this again later for the created global.

You may use the System Manager’s Modify System Parameters (MSP) utility to define the initial $ZU(23) setting for processes and to define the character set and string collation switch associated with a particular directory. The character set and string collation switch of a directory are used as the default values for newly created globals in the directory. Also, when a process switches to a new directory using $ZU(5), the $ZU(23) setting for the process is set to the new directory’s character set and string collation switch.

You may also use the MSP utility to define a system-wide default for the character set and string collation switch.

To avoid inconsistent results from SQL queries on a database, you must define the collation sequences for all of the globals in the database as well as $ZU(23) to be identical.

Changing Collation Sequence on Non-ISM Systems

If you are running Open M/SQL on a non-ISM host M system, please consult your M vendor's system management guide for information on changing the M collating sequence.

%NOCHECK

InterSystems' SQL supports the use of the %NOCHECK keyword in SQL INSERT, UPDATE, and DELETE statements to suppress the following validation checking:

- Field Validation Code
- Base Table Validation Code
- Checking for required fields
- Checking for field uniqueness

%NOCHECK does not suppress External-to-Internal Conversion Code nor does it suppress the execution of filing triggers when appropriate.

The syntax for using the %NOCHECK keyword is:

```
INSERT %NOCHECK ...

UPDATE %NOCHECK ...

DELETE %NOCHECK ...
```

Use of the %NOCHECK keyword is appropriate in an application that reads rows from one table and writes them directly to another. If the validation and conversion code is the same for both tables, there is no need to check a row that is read from the first table before filing it to the second.

Similarly, you may want an application to do its own validation checking and conversion, and bypass the code built into the Data Dictionary to avoid redundant checking.

SQL Transaction Processing

ANSI-Standard SQL supports two transaction processing statements, COMMIT and ROLLBACK.

In ANSI-Standard SQL, unlike Open M/SQL, every operation (SELECT, UPDATE, etc.) automatically begins a transaction if one is not already in progress.

InterSystems' SQL gives explicit control of transactions to the application programmer and provides two extension keywords for this purpose:

- %BEGTRANS — begins a transaction
- %INTRANS — sets the variable SQLCODE to one of the following values:

Return Value	Meaning
0	Transaction not in progress
-1	Transaction in progress (but journalling was not enabled when the transaction started)
>0	Transaction in progress (and journalling was enabled when the transaction started)

In order to perform a ROLLBACK operation, journalling must be enabled for the system (i.e., %INTRANS must return a non-negative value) as well as for every global whose modifications are to be considered part of the transaction (i.e., whose modifications need to be rolled back as part of the ROLLBACK operation).

The following example of embedded SQL uses the %INTRANS keyword to determine whether a transaction is in progress and, if not, begins one:

```
new oldtrans
&sql(%INTRANS) set oldtrans=SQLCODE
if 'oldtrans &sql(%BEGTRANS)
...
if failed &sql(ROLLBACK) go done
...
if 'oldtrans go done
&sql(COMMIT)
done  ...
```

If a transaction was not previously in progress, this code begins a new one and commits it at the end. If a transaction already was in progress, this code does not commit it. If an error occurs during processing, the code performs a rollback instead of a commit.

Transactions include not only SQL modifications but also any direct global sets and kills performed by a M program.

Open M/SQL supports SQL transaction processing only on those platforms that support M transaction processing. Currently, Open M/SQL supports transaction processing when running on the following host M systems:

- ISM
- DSM

Privilege Operators

InterSystems' SQL supports the SQL keyword USER in accordance with ANSI-Standard SQL.

InterSystems' SQL also supports the SQL GRANT and REVOKE statements, as defined in ANSI-Standard SQL, with several extensions specific to Open M/SQL.

ANSI-Standard SQL permits the granting and revoking of privileges only on base tables. InterSystems' SQL extends ANSI-Standard by allowing the granting and revoking of privileges on all of the following objects:

- Base Tables
- Views
- Forms
- Menu Objects
- Menus (Old-Style)
- Reports
- Queries

GRANT

You may use the SQL GRANT statement to grant access privileges on specified objects to Open M/SQL users. The GRANT statement has an option to allow the user to grant access to other users.

Note The owner of an object automatically holds all privileges on that object.

The GRANT statement uses the following syntax:

```
GRANT <list of actions>
ON [<object type>]<object name>
TO <grantees> [WITH GRANT OPTION]
```

For example:

```
GRANT %ALTER, SELECT, INSERT
ON Patients
TO Chris WITH GRANT OPTION
```

The table below describes the elements of the GRANT statement syntax:

Table 9-4: GRANT Statement Syntax Elements

Syntax Element	Meaning
<list of actions>	Here you specify a list of Open M/SQL privileges actions, delimited by commas. You may specify an asterisk (*) or ALL PRIVILEGES to indicate privileges on all actions, or you may list one or more of the privilege actions shown in the table below.
<object type>	Here you specify the type of object on which you want to grant privileges. The default object type is base table. Omit this parameter to specify base table as the object type. Alternatively, you may specify any of the following object types: ▪ %FORM ▪ %MENUOBJECT ▪ %MENU ▪ %REPORT ▪ %QUERY You may grant privileges only on one object type at a time. **Note:** A view is also a valid object type on which privileges may be granted. SQL treats views as tables.
<object name>	Specify the name(s) of the specific object(s) on which you want to grant privileges, or an asterisk (*) to indicate all object names of the specified object type. If you specify a list of individual objects, you must delimit the object names by commas. You may grant privileges on multiple objects at once (though they must all be the same object type).
<grantees>	Specify IDs for the user(s) to whom you wish to grant privileges. These IDs may be any of the following: ▪ Role name(s) ▪ UserName(s) of user(s) registered in the User table ▪ UserName(s) of user(s) not registered in the User table You may also enter an asterisk (*) or *_PUBLIC* to grant privileges to all users. When you do this, Open M/SQL assigns the privileges to the UserName "_PUBLIC", which means unrestricted access for all users.
[WITH GRANT OPTION]	You may optionally specify the WITH GRANT OPTION keyword to allow the user(s) to whom you are granting these privileges to grant the same privileges to other users.

The table below lists and describes the privilege actions you can grant to or revoke from Open M/SQL users on Open M/SQL objects:

Table 9-5: Open M/SQL Privilege Actions

Privilege Action	Meaning
%ALTER	Privileges to modify the definition of an object
SELECT	Privileges to retrieve information from a base table or view, or to run an object such as a form
INSERT	Privileges to insert a row
UPDATE	Privileges to update an existing row
DELETE	Privileges to delete an existing row
REFERENCES	Privileges to create designative references to a table

REVOKE

You may use the SQL REVOKE statement to revoke access privileges on specified objects from Open M/SQL users. Only the grantor of privileges has the authority to revoke those privileges.

The SQL REVOKE statement uses the following syntax:

```
REVOKE [GRANT OPTION FOR] <list of actions>
ON [<object type>] <object name>
FROM <grantees> [CASCADE]
```

For example:

```
REVOKE %ALTER, SELECT, INSERT
ON %FORM PatientInfo
FROM Chris CASCADE
```

The <*list of actions*>, <*object type*>, <*object name*>, and <*grantees*> syntax elements have the same meaning in the REVOKE statement as they do in the GRANT statement.

The table below describes the two optional keywords supported by the REVOKE statement:

Table 9-6: REVOKE Statement Syntax Elements

Keyword	Meaning
[GRANT OPTION FOR]	You may optionally specify this keyword to revoke only the GRANT option (the user's ability to grant the privileges to other users) for the specified privileges and not the privileges themselves.
[CASCADE]	You may optionally specify this keyword to strip the privileges on the specified object from all users who received them from the revokee (or via the revokee). For example, if you revoke privileges from user A, and user A had previously granted those same privileges to user B, then under the Cascade option you also revoke those privileges from user B (as well as from any users to whom user B may have granted the privileges).

%CHECKPRIV Keyword

The %CHECKPRIV keyword is an InterSystems SQL extension that allows an Open M/SQL user to check whether or not s/he holds a specified privilege on a specified Open M/SQL object.

The syntax for using %CHECKPRIV is:

```
%CHECKPRIV [GRANT OPTION FOR] <action>
ON <object type> <object name>
```

For example:

```
%CHECKPRIV %ALTER
ON %FORM PatientInformation
```

If the user holds the specified privilege, the query returns an SQLCODE value of 0 (success). If the user does not hold the specified privilege, the query returns an SQLCODE value of 100.

%CHECKPRIV may check only one action on only one object at a time.

The <*action*>, <*object type*>, and <*object name*> syntax elements have the same meaning in the %CHECKPRIV statement as they do in the GRANT and REVOKE statements.

The GRANT OPTION FOR keyword is optional. If you specify this keyword, the query checks whether or not the user holds the GRANT privilege on the specified privilege— not whether or not the user holds the specified privilege itself.

In order to run a %CHECKPRIV query, you must ensure that an Authorization ID is established prior to executing the query. You can establish an Authorization in any of the following ways:

- By executing the query from any of the following locations within Open M/SQL:
  - Interactive Query Editor
  - A menu option of action type SQL
  - A trigger item of action type SQL

  When Open M/SQL is running, the Authorization ID is always defined.
- By issuing an M command with the following syntax prior to running the query:

  ```
  > do setaid^%msql("<UserName>")
  ```

  where <*UserName*> is a UserName that is registered in the User Table.

  For example:

  ```
  > do setaid^%msql("Zeus")
  ```

SQLCODE Values

After a %CHECKPRIV query, the SQLCODE variable has the value 0 if the user holds the tested privilege, or 100 if the user does not hold the tested privilege.

GRANT and REVOKE queries set the SQLCODE variable to 0 when they successfully complete the intended operation. If a GRANT or REVOKE query is inapplicable because a user already held (or did not hold in the case of REVOKE) the specified privileges, the query sets SQLCODE to 100.

For a complete listing of other SQL error messages and their meanings, refer to Appendix A: *SQL Error Messages*.

Using Subqueries

InterSystems' SQL allows the use of subqueries in accordance with ANSI-Standard SQL.

A subquery is an SQL SELECT statement query embedded within another SQL SELECT statement query.

According to ANSI-Standard SQL, a subquery must be embedded within a WHERE clause or a FROM clause. When embedded within a WHERE clause, the result of the subquery must be an atomic data value (one column, aggregate, or expression) or a truth value. When embedded within a FROM clause, the result of the subquery is a virtual table that may consist of many columns and many rows.

In addition to permitting subqueries in WHERE clauses and FROM clause, InterSystems' SQL also permits subqueries to be embedded within the SELECT clauses.

You may nest all subqueries may to any number of levels.

Subqueries normally appear as complete queries enclosed within parentheses in a WHERE clause. An SQL query that has the form:

```
SELECT...
FROM...
WHERE...
```

may contain a subquery in the WHERE clause, thus the form:

```
SELECT...
FROM...
WHERE... (SELECT... FROM... WHERE...)
```

Using a Subquery in a WHERE Clause

The set of rows yielded by a subquery is used to restrict the outside query. Below are some examples of various ways to use subqueries in logical conditions within WHERE clauses.

Note Currently, Open M/SQL supports only correlated subqueries—queries in which the result of the subquery depends on the value of the specific row of the outer query. Non-correlated subqueries, in which the outer and inner queries are completely independent, are not supported at this time.

Expression Matches Some Value in Subquery Output

In the example:

```
WHERE <expr> IN (SELECT... FROM... WHERE...)
```

the subquery retrieves one column for some set of rows, and the predicate is the truth value of <*expr*> being equal to the data value of the column for any of the retrieved rows.

Expression Does Not Match Any Value in Subquery Output

In the example:

```
WHERE <expr> NOT IN (SELECT... FROM... WHERE...)
```

the subquery retrieves one column from some set of rows, and the predicate is the truth value of <*expr*> NOT being equal to the data value of the column for any of the retrieved rows.

Subquery Retrieves At Least One Row

In the example:

```
WHERE EXISTS (SELECT... FROM... WHERE...)
```

the predicate tests for the existence of one or more rows specified by the subquery. Typically, the subquery takes the form:

```
SELECT *
FROM...
WHERE...
```

Expression Compares With Values in Subquery Output

In the example:

```
WHERE <expr> <Compar-op> (SELECT... FROM... WHERE...)
```

the subquery must retrieve an atomic value (one column or aggregate), and the predicate is the truth value of the comparison operation of <*expr*> with rows from the subquery output.

Expression Compares with Some Values in Subquery Output

In the example:

```
WHERE <expr> <Compar-op> SOME (SELECT... FROM... WHERE...)
```

the subquery retrieves some set of rows, and the predicate tests the truth value of the comparison operation of <*expr*> with *any* row from the subquery output.

Expression Compares with All Values in Subquery Output

In the example:

```
WHERE <expr> <Compar-op> ALL (SELECT... FROM... WHERE...)
```

the subquery retrieves some set of rows, and the predicate tests the truth value of the comparison operation of <*expr*> with *all* rows from the subquery output.

Using a Subquery in a FROM clause

When you use a subquery in a FROM clause, the results of the subquery define the virtual table on which the outer query is based. This virtual table may consist of many columns and many rows.

You must observe the following restrictions when defining subqueries in the FROM clause:

1. You may not put anything else in the FROM clause of the outer query—the subquery must be alone
2. The subquery may not have a subquery in its FROM clause.

   However, it may have subqueries anywhere else (in its WHERE clause or in its SELECT clause), and those subqueries may have subqueries anywhere (including in their FROM clauses).
3. The subquery may not contain aggregates or DISTINCT.

Note The use of query-based views in a FROM clause is equivalent to specifying the view's query as a subquery in the FROM clause. Therefore, the above restrictions also apply to query-based views. For more information on query-based views, see the section of this chapter entitled "Query-Based Views" on page 9-57.

Open M/SQL Subquery Extensions

InterSystems' SQL provides two extensions to the ANSI-Standard SQL specifications for subqueries:

1. While ANSI-Standard SQL permits subqueries to be embedded only within WHERE clauses, Open M/SQL adds the ability to embed subqueries within SELECT clauses, in the form shown below:

```
SELECT  (SELECT... FROM... WHERE...)
FROM...
WHERE...
```

2. InterSystems' SQL supports two logical operators, FOR SOME and FOR ALL, which enable you to invoke subqueries more succinctly.

Subquery Embedded in SELECT Clause

Subqueries embedded within a SELECT clause are scalar subqueries—they always return one scalar value.

The following SQL query contains a scalar subquery embedded in the SELECT clause:

```
SELECT Snum,Sname,Scity,(SELECT Count(Pnum)
   FROM Parts WHERE Suppliers.Scity=Parts.Pcity)
FROM Suppliers
```

This query yields the following output table:

Snum	Sname	Scity	Count
S1	Klein	Provo	1
S2	James	Daytona	
S3	Travers	Boston	1
S4	Martin	Provo	1

In the following examples, note the use of the "patientnum" field, which is the Row ID of the "Patients" table and the parent reference within the child table "Tests".

FOR ALL Operator

The FOR ALL logical operator tests whether every value of one expression satisfies the condition(s) of a second embedded expression. Essentially, FOR ALL provides a shorthand version of a subquery.

FOR ALL uses the following syntax:

```
FOR ALL expr1(expr2)
```

For example, the following query uses the FOR ALL operator:

```
&sql(SELECT PatientName,Age
   FROM Patients
   WHERE FOR ALL tests (tests.Result="Positive"
      AND patients.patientnum=tests.patientnum))
```

and is equivalent to:

```
&sql(SELECT PatientName,Age
   FROM Patients
   WHERE NOT EXISTS (SELECT * FROM Tests
      WHERE NOT (tests.Result = "Positive"
      AND patients.patientnum = test.patientnum)))
```

FOR SOME Operator

The FOR SOME logical operator tests whether at least one value of one expression satisfies the condition(s) of a second embedded expression. Essentially, FOR SOME provides a shorthand version of a subquery.

FOR SOME uses the following syntax:

```
FOR SOME expr1(expr2)
```

For example, the following query uses the FOR SOME operator:

```
&sql(SELECT PatientName,Age
FROM Patients
WHERE FOR SOME Tests (Tests.patientnum=Tests.patientnum
   AND Tests.Result="Positive"))
```

and is equivalent to:

```
&sql(SELECT PatientName,Age
   FROM Patients
   WHERE patientnum =SOME (SELECT patientnum
      FROM Tests
      WHERE Tests.Result="Positive"))
```

Query-Based Views

InterSystems' SQL supports the following SQL DDL statements for use in creating, editing, and deleting query-based views:

- CREATE VIEW
- ALTER VIEW
- DROP VIEW

When you create a query-based view, you are explicitly defining the query that joins the base table in the view.

You may issue SQL queries using these DDL statements in either of two places:

- Via the Interactive Query Editor
- As embedded SQL in M macro source code

You may compile these SQL statements in any directory with the same result. Open M/SQL performs no referential integrity checking on these SQL statements at compile time. It checks referential integrity only at run time.

CREATE VIEW

The CREATE VIEW statement uses the following syntax:

```
CREATE VIEW <viewname> AS SELECT...FROM...WHERE...
```

For example:

```
CREATE VIEW Outstanding_Charges AS
SELECT Patient_Name, Date, Amount_Due FROM Billing
WHERE (Billing.Lab_Charges!="Paid"
   OR (Billing.Office_Visit_Charges!="Paid")
```

ALTER VIEW

The ALTER VIEW statement uses the following syntax:

```
ALTER VIEW <viewname> AS SELECT...FROM...WHERE...
```

ALTER VIEW uses exactly the same syntax as CREATE VIEW.

When you use ALTER VIEW, you delete and recreate the entire view, retaining the same view name and internal view number.

DROP VIEW

The DROP VIEW statement uses the following syntax:

```
DROP VIEW <viewname>
```

Restrictions on Defining Query-Based Views

You must observe the following restrictions when defining query-based views:

- The query may not contain aggregates or DISTINCT.
- The query may not itself reference another query-based view.

You May Name Query-Based Views in FROM Clause

InterSystems' SQL allows you to name query-based views in the FROM clause of SQL SELECT statement queries.

However, a query that names a query-based view in its FROM clause cannot name any additional items in the FROM clause.

Furthermore, a query-based view cannot appear in the FROM clause of a query as report data source.

CHAPTER

10

Query Generation and Processing

Open M/SQL provides two facilities for creating SQL queries on an ad hoc basis, the Query Generator and the Interactive Query Editor.

This chapter describes how to use these query generation facilities to create, run save, and copy queries.

Specifically, it covers the following topics:

- Facilities for Creating SQL Queries
- Using the Query Generator
- Using the Interactive Query Editor
- Running Queries
- ASCII-Delimited Output for Queries
- Copying Queries
- List Queries Report
- Detailed Query Listing Report

Facilities for Creating SQL Queries

The table below summarizes the capabilities and relative advantages of the two Open M/SQL query generation facilities:

Table 10-1: Query Generation Facilities

Facility	Description	Advantages
Query Generator	Provides an easy-to-use template of SQL SELECT statement syntax equipped with fields for the appropriate SQL clauses, including SELECT, FROM, WHERE, ORDER BY, GROUP BY, and HAVING.	You can define a SELECT query by simply filling in the template. You do not need to know SQL syntax rules.
Interactive Query Editor	Provides a free-form SQL editor environment (similar to the Full Screen Editor) where you can create and run any valid SQL query. Provides full screen editing capabilities.	You can define any valid SQL query, making the Interactive Query Editor more flexible than the Query Generator.

Both the Query Generator and the Interactive Query Editor can compile the finished query and execute it on your command.

The output of SELECT-statement queries appears in standard tabular report format. You may print this output in any of the following ways:

- To your screen
- To a hard copy printer
- To a file in ASCII-delimited output format

Accessing the Query Generation Facilities

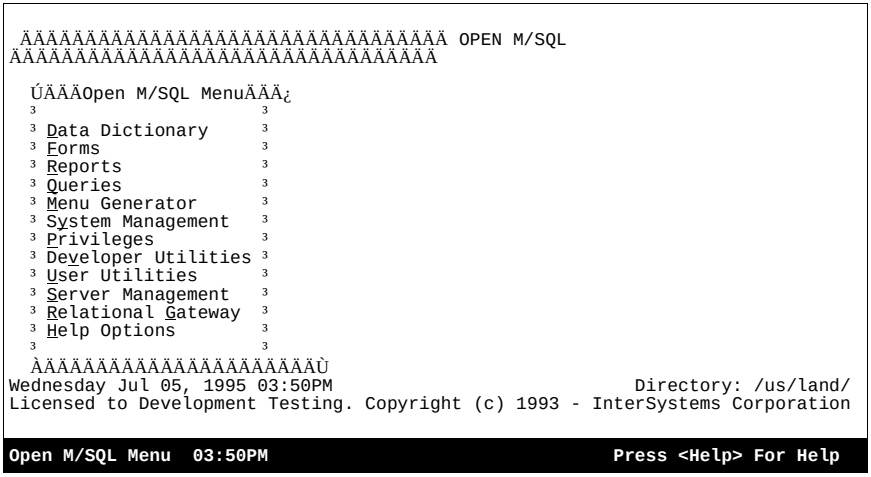
The Open M/SQL query generation facilities (Query Generator and Interactive Query Editor) as well as several query management utilities are available as options on the SQL menu.

Procedure **To access the SQL menu:**

1. **At the M programmer prompt, type the following command to enter Open M/SQL:**

```
> do ^%msql
```

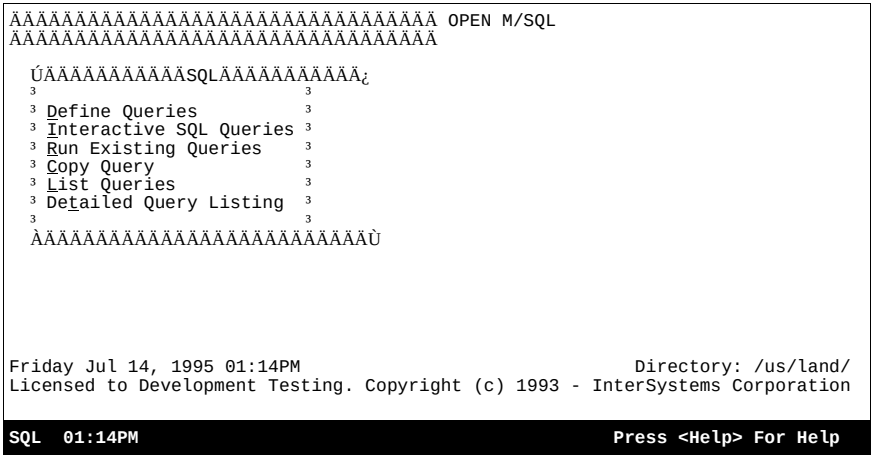
You see the Open M/SQL Main Menu, as shown below:



2. From the Open M/SQL Main Menu, select the Queries option.

Note: You may type *q* to select this option—it is a mnemonic accelerator.

You see the SQL Menu, as shown below:



The table below lists and describes the options on the SQL menu:

Table 10-2: Options on SQL Menu

Option	Description
Define Queries	Select this option to access the Query Generator, which provides a template for defining SQL SELECT statement queries.
Interactive SQL Queries	Select this option to access the Interactive Query Editor, which provides a full screen editing environment for creating and executing SQL statements using any valid syntax.
Run Existing Queries	Select this option to access the Run Existing Queries utility, which allows you to select any query defined via the Query Generator or Interactive Query Editor and run it.
Copy Query	Select this option to access the Copy Query utility, which allows you to select any query defined via the Query Generator or Interactive Query Editor and copy its definition to create a new query.
List Queries	Select this option to print the List Queries report, which displays an alphabetical listing of all queries defined within the current database.
Detailed Query Listing	Select this option to print the Detailed Query Listing report, which displays various information about one or more specified queries.

Using the Query Generator

The Query Generator provides an easy-to-use template for defining SQL SELECT statement queries. The template is equipped with fields for the appropriate SQL clauses, including SELECT, FROM, WHERE, ORDER BY, GROUP BY, and HAVING. This enables you to create a SELECT query by simply filling in the template.

The Query Generator automatically generates your query as a cursor-based SELECT statement query, meaning it can retrieve multiple data rows into the output table.

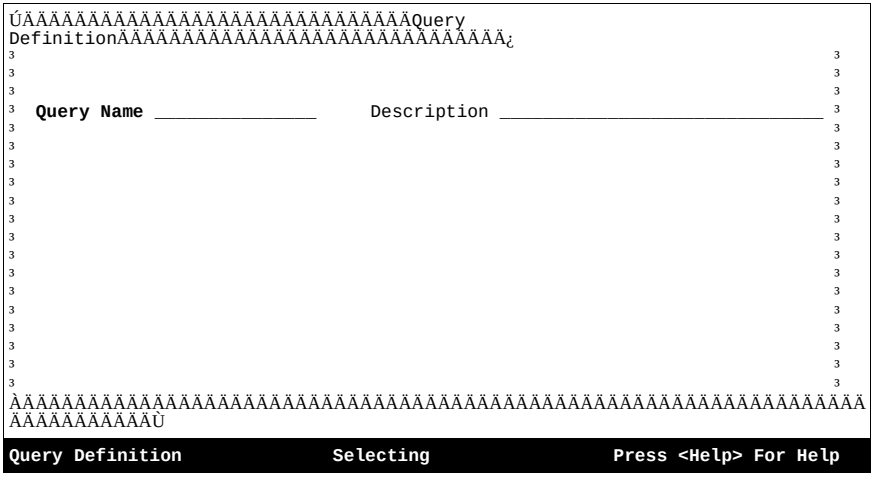
Since the Query Generator automatically generates the cursor-based SELECT statement syntax, you do **not** need to use any of the following SQL commands:

- DECLARE CURSOR
- OPEN CURSOR
- FETCH CURSOR
- CLOSE CURSOR
- INTO

Procedure **To define a SELECT statement query using the Query Generator:**

1. **From the SQL Menu, select the Define Queries option.**

   You see the Query Definition lookup window, as shown below:



2. **At the Query Name field in the Query Definition lookup window, enter a query name, and press <RETURN>.**

3. **At the Query Description field, enter a query description, and press <RETURN>.**

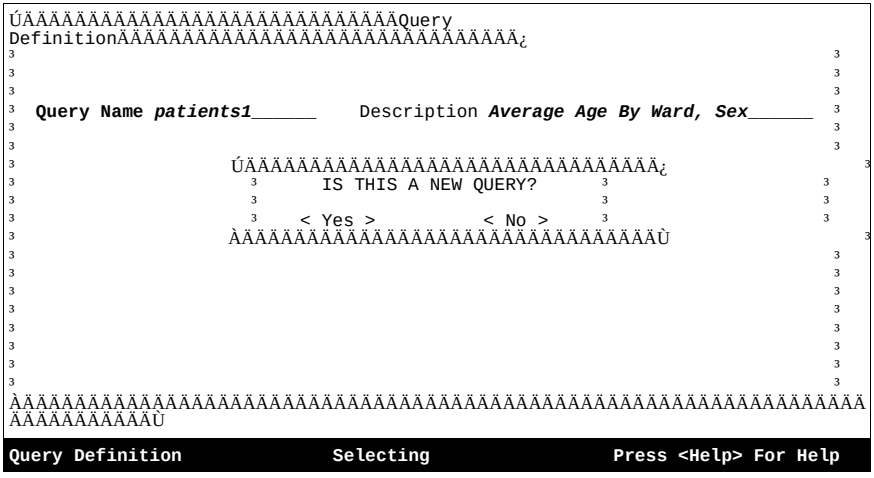
   You can retrieve an existing query to edit or create a new one.

   a. To retrieve an existing query, enter a complete or partial query name in the Query Name field and/or a complete or partial query description in the Query Description field, and press <RETURN>. You see a lookup box that lists all matching entries.

   To see a lookup box that lists all queries defined in the current database, leave the Query Name and Query Description fields blank and press the <SEARCH CURRENT TABLE> key. Use the cursor positioning keys to navigate within the lookup box, and press <RETURN> to select an entry.

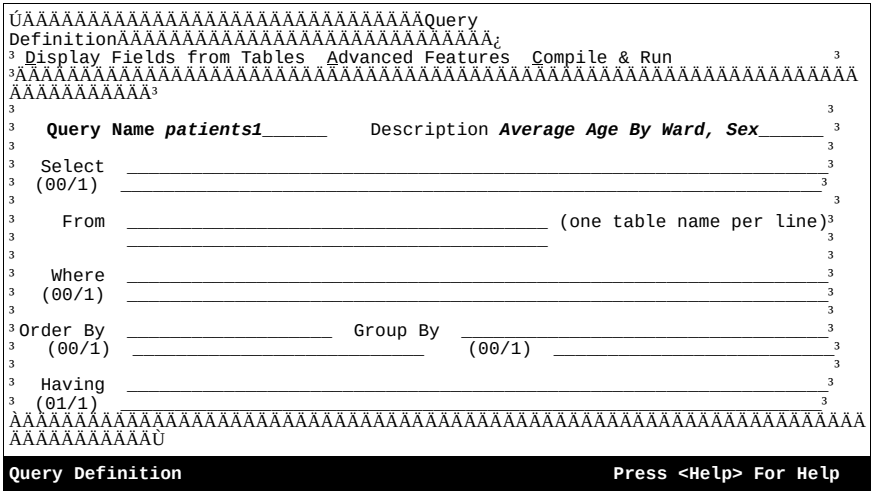
   b. To create a new query, enter a new query name (it must not match the name of any existing query) in the Query Name field, optionally enter a query description in the Query Description field, and press <RETURN>.

   You see the "IS THIS A NEW QUERY?" dialog box, as shown below:.



   Here, you may create a new query by pressing <RETURN> on the <Yes> action field.

When you have entered a query name (either new or existing), the Query Definition window fills out with the full template for defining a SELECT statement query:.



The table below lists and describes the fields located on the Query Definition window's SELECT statement template:

Table 10-3: Fields on Query Definition Window

Field	Description
Query Name	This is the name that identifies the query. The name may range from 1 to 30 characters in length and may include any alphanumeric characters, including the underscore character. The name must always begin with an alphabetic character. This is a required field.
Query Description	Here you may enter a brief description for the query. The description may be up to 60 characters in length and may include any alphanumeric characters, including underscores and blank spaces.
SELECT	Here you may enter one or more expressions, separated by commas, that specify the data columns to appear in the query output. The expressions may contain field names, aggregate functions, and the InterSystems' SQL extension keywords %FOREACH, %AFTERHAVING, and DISTINCT BY. (For more information on these and other SQL extensions, see Chapter 9, *Open M/SQL Implementation of SQL*). The SELECT clause may also contain subqueries. All field names specified in the SELECT clause must originate in a base table or view that is listed in the From clause. Each line of this field scrolls to right and can accommodate up to 250 characters. If you continue the expression onto the next line, you must end the previous line with a comma.

Table 10-3: Fields on Query Definition Window (Continued)

Field	Description
FROM	Here you may specify the data source of the query. You may specify one or more base tables or views as the data source, or you may specify one query-based view. You may select only base tables/views that are defined in the current database. To see a list of all base tables/views defined in the current database, you may press the <LIST CHOICES> key. Each new base table or view name you enter must occupy a separate line. After a base table name you may optionally specify an alias for the base table, separated by a space from the base table name.
WHERE	Here you may enter one or more conditional expression(s) that qualifies or disqualifies specific rows from the query output. You must link multiple expressions together by AND or OR. Use this clause to restrict the query's data capture to very specific sets of information. The WHERE clause accepts all comparison operators and the BETWEEN, LIKE, NULL, IN, EXISTS, FOR ALL, and FOR SOME predicates, but may not contain any aggregate functions. You may also embed subqueries in the WHERE clause.
ORDER BY	Here you may specify one or more field(s) or expression(s), separated by commas, to designate the sort order for rows in the query output. You may specify field(s) and/or expression(s) that do not appear in the SELECT clause, as well as those that do. You may reference a field either by name or by its ordinal number in the SELECT clause, if it appears there. If you specify multiple items, each successive item has lesser precedence in the sort evaluation. Sorts may be performed in ascending (the default) or descending order, as specified **Note:** If you enter one ORDER BY field name per line, it is not necessary to enter a comma at the end of each line.
GROUP BY	Here you may specify one or more fields, separated by commas, that will be used to break up the final query output into groups. For each distinct group, the query will return only one row. For example, if you group by the field “sex”, and “sex” contains only the two values “Male” and “Female”, the query will discern two groups and output only one row for each of them. In such case, a name field listed in the SELECT clause is rendered not meaningful, as the query will output only one name corresponding to each value of “sex”. The GROUP BY clause is conceptually similar to the Open M/SQL extension %FOREACH, but GROUP BY operates on an entire query, while %FOREACH allows selection of aggregates on sub-populations without restricting the entire query population. **Note:** If you enter one GROUP BY field name per line, it is not necessary to enter a comma at the end of each line.

Table 10-3: Fields on Query Definition Window (Continued)

Field	Description
HAVING	Here you may enter a conditional expression that determines whether or not a given group should be included in the query output. This field enables you to set restrictions on data groups. The HAVING clause operates on groups of data in much the same fashion as the WHERE clause operates on individual rows of data. By specifying the HAVING clause, you can effectively isolate very specific sets of information. Unlike the WHERE clause, the HAVING clause may contain aggregate functions.

4. **Enter the appropriate text into the fields of the Query Definition template to define a SELECT-statement query that queries the database for the information you are seeking.**

5. **When you have finished defining your query, press the <PROCEED> key to save your query definition and exit the Query Definition template.**

   You return to a blank Query Definition lookup window, where you may create or select another query.

6. **To return to the SQL menu, press the <PREVIOUS> key.**

Example

Below is a sample query defined in the Query Definition template:.

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄQuery
DefinitionÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ Display Fields from Tables  Advanced Features  Compile & Run           ³
³ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄ³
³                                                                        ³
³   Query Name patients1______    Description Average Age By Ward, Sex______ ³
³                                                                        ³
³   Select  ward,sex,AVG(age)___________________________________________³
³   (00/1)  ____________________________________________________________³
³                                                                        ³
³     From  Patients_______________________________ (one table name per line)³
³           _______________________________________                      ³
³                                                                        ³
³    Where  ____________________________________________________________³
³   (00/1)  ____________________________________________________________³
³                                                                        ³
³ Order By  ward,sex___________________  Group By  ward,sex__________________³
³   (00/1)  ___________________________    (00/1)  __________________________³
³                                                                        ³
³   Having  AVG(age) < 62_______________________________________________³
³   (01/1)  ____________________________________________________________³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
Query Definition                                    Press <Help> For Help
```

This query considers all patients in the “Patients” table. It yields groups that correspond to each sex within each ward where the average age of that group is less than 62. It orders the output rows first by ward, then by sex.

The query shown in the Query Definition template above can be written out as follows:

```
SELECT ward,sex,AVG(age)
   FROM Patients
   GROUP BY ward,sex
   ORDER BY ward,sex
   HAVING AVG(age) < 62
```

The following table shows a sample output for this query. Each column represents a SELECT clause expression, and each row represents a distinct GROUP BY row:

Table 10-4: Sample Query Output

Ward	Sex	Average Age
1	F	43.53
1	M	47.21
2	F	17.33
2	M	18.10
4	F	59.69
6	F	35.46
6	M	36.71

This example omits wards 3 and 5 because both sex groups within each ward fail to satisfy the condition in the HAVING clause, i.e. the average age of each group is greater than 62. The table omits the row corresponding to “Male” in ward 4 for the same reason.

Note ANSI-Standard SQL does not support column title control and other output formatting features. Therefore, query output will not appear as neatly formatted as shown in the above example. To query the database and produce neatly formatted reports, you should the Open M/SQL M/PACT report writer tool.

Query Definition Menu Bar

Located at the top of the Query Definition template is the Query Definition menu bar, which contains options that provide additional functionality to the Query Generator environment.

To access the Query Definition menu bar, press the <ENHANCE> key twice from within the Query Definition main window.

To select an option from the Query Definition menu bar, use the <RIGHT ARROW> and <LEFT ARROW> keys to position the select bar on a desired option and press <RETURN>, or type the highlighted letter in the name of a desired option.

The table below lists and describes the three options on the Query Definition menu bar:

Table 10-5: Options on Query Definition Menu Bar

Name	Description
Display Fields from Tables	Select this option to see a list that displays all fields located on all base tables and views specified in the FROM clause of the query
Advanced Features	Select this option to access the Query Definition Advanced Features window, which displays various information about the query and allows you to define parameters for printing the query in ASCII-delimited output format.
Compile & Run	Select this option to compile and run the query

Displaying Fields From Tables and Views

You may use the Display Fields from Tables option on the Query Definition menu bar to see a list of all fields located on all base tables and views specified in the query's FROM clause.

The list displays all fields located on the table/view specified on the first line of the FROM clause followed by all fields on the table/view specified on the second line of the FROM clause, and so on.

This list may help you identify which fields are available for use in other clauses of the query.

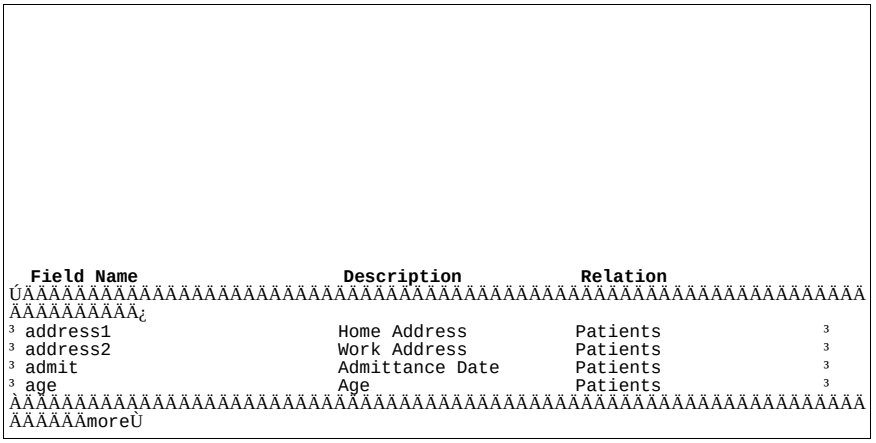
The list is for informational purposes only; you cannot select items from it.

Procedure **To display a list of all fields located on the base tables/views specified in the FROM clause:**

1. **From the Query Definition main window, press the <ENHANCE> key twice to access the Query Definition menu bar.**

2. **From the Query Definition menu bar, press <RETURN> to select the Display Fields from Tables option.**

   You see a display similar to the following:



3. **Use the <UP ARROW> and <DOWN ARROW> keys to scroll up or down in order to see more field names.**

   The tag “more” appears in the bottom right-hand corner of the display box to indicate the presence of more fields below and in the upper right-hand corner of the display box to indicate the presence of more fields above.

   Note: This option does not list fields from designated tables that are pointed to by Designative Reference fields within the listed base tables. You may, however, reference such fields within the query using proper join (arrow) syntax. For more information on how to do this, refer to the section entitled “Implicit Joins” in Chapter 9, *Open M/SQL Implementation of SQL*.

4. **Press the <PREVIOUS> key to return to the Query Definition menu bar.**

Query Definition Advanced Features

You may use the Advanced Features option on the Query Definition menu bar to access the Query Definition Advanced Features window.

The Query Definition Advanced Features window displays the following information about your query:

- Owner (this value is read-only)
- Run-Time Measure (this value is read-only)
- Routine Name (this value is modifiable)

The Query Definition Advanced Features window also lets you define parameters for printing your query in ASCII-delimited output format.

Open M/SQL supports the output of queries to ASCII-Delimited text files, which are formatted for import into many PC software packages.

Procedure **To access and use the Query Definition Advanced Features window:**

1. **From the Query Definition main window, press the <ENHANCE> key twice to access the Query Definition menu bar.**

2. **From the Query Definition menu bar, type *a* to select the Advanced Features option.**

   You see the Query Definition Advanced Features window, as shown below:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄQuery
DefinitionÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ Display Fields from Tables  Advanced Features  Compile & Run          ³
³ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄ³
³                                                                        ³
³  Query Name patients1______     DescÚÄÄÄÄÄÄÄÄÄÄÄAdvanced
FeaturesÄÄÄÄÄÄÄÄÄÄ¿
³                                     ³                                ³
³  Select  ward,sex,AVG(age)__________³ Query Owner       Run Time Measure  ³
³  (00/1)  ___________________________³ _SYSTEM               1195          ³
³                                     ³                                ³
³    From  Patients___________________³ Routine Name mq9____           ³
³          ___________________________³                                ³
³                                     ³ ASCII-Delimited Files Only? No__  ³
³   Where  ___________________________³                                ³
³  (00/1)
___________________________³ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ³
³                                     ³                                ³
³Order By  ward,sex___________________³        Quote Character "________  ³
³  (00/1)  ___________________________³                                ³
³                                     ³  End Of Field Delimiter ,________  ³
³  Having  AVG(age) < 62______________³                                ³
³  (00/1)  ___________________________³ End Of Record Delimiter 13,10____  ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÙ
Advanced Features                                        Press <Help> For Help
```

The following table lists and describes the fields located on the Query Definition Advanced Features window:

Table 10-6: Fields on Query Definition Advanced Features Window

Field	Description
Query Owner	This field shows the UserName of the user who created (and therefore owns) the query. You cannot edit this field.
Run Time Measure	This field provides a relative measure of the system cost associated with running the query. The number shown is based on the complexity of both the query and the database. Typically, the larger the number, the more slowly the query executes. Open M/SQL calculates the run-time measure when you compile the query and recalculates it every time you recompile. This value is read-only and cannot be altered in any way.
Routine Name	This field displays the prefix for all routine names generated by the query. Open M/SQL assigns a default prefix, such as “mq9” in the above example. You may optionally override the default prefix by replacing it with a string of 1 to 7 alphanumeric characters (first character must be alphabetic) to serve as the override prefix. All routines generated for the query consist of this prefix followed by a single character (1-9, then A-Z, then a-z), which is appended by the compiler. Specifying your own routine prefix may help you to remember it when attempting to call the query directly from M using the query call syntax, as described in the section of this chapter entitled “Running a Query from M Code” on page 10-31.
ASCII-Delimited Files Only?	Here you may specify ASCII-Delimited as the only allowable output format for the query, i.e. no printed format is allowed. Answer *Yes* to specify ASCII-Delimited as the only allowable output format for the query. This enables you to download an unlimited number of fields to the ASCII-Delimited file. Answer *No* to not restrict output of the query to ASCII-delimited format only. If you answer *No*, you can still send query output to an ASCII-Delimited file, but the number of fields you may select is limited to the number of fields supported by a standard printed query. *No* is the default response. For more information on sending query output to ASCII-delimited files, see the section of this chapter entitled “ASCII-Delimited Output for Queries” on page 10-36.
Quote Character	Here you may specify the character(s) used to enclose field values when using ASCII-delimited output format. You may specify any text of ten characters or less. The default is the double quote character (“).

Table 10-6: Fields on Query Definition Advanced Features Window

Field	Description
End of Field Delimiter	Here you may specify the character(s) used to act as the separator between field values when using ASCII-delimited output format. You may specify any text of ten characters or less. The default is the comma character (,).
End of Record Delimiter	Here you may specify the character(s) used to act as the separator between records (rows) when using ASCII-delimited output format. You may specify a list of ASCII values (numbers) separated by commas. The default value is the ASCII sequence 13,10 (same as <CR><LF>).

3. **At the Routine Name field, you may optionally replace the default routine prefix assigned to your query by Open M/SQL with an override routine prefix.**

   The routine prefix may be 1 to 7 characters in length and may include any alphanumeric characters, except for the first character, which must be alphabetic.

4. **At the ASCII-Delimited Files Only? field, answer *Yes* or *No* to indicate whether or not you want to restrict the query to ASCII-Delimited output format only.**

   No is the default response.

5. **At the Quote Character field, specify the character(s) used to enclose field values when using ASCII-delimited output format.**

   The default is the double quote character (“).

6. **At the End of Field Delimiter field, specify the character(s) used to act as the separator between field values when using ASCII-delimited output format.**

   The default is the comma character (,).

7. **At the End of Record Delimiter Field, specify the character(s) used to act as the separator between records (rows) when using ASCII-delimited output format.**

   The default value is the ASCII sequence 13,10 (same as <CR><LF>).

8. **When you complete your work in the Query Definition Advanced Features window, press the <PROCEED> key to save your definitions and exit.**

Compile and Run the Query

You may use the Compile & Run option on the Query Definition menu bar to compile and run your query.

When you select this option, you see the following prompt at the bottom of the screen:

```
Parse, Optimize and Compile in the Background? No__
```

Press <RETURN> at this prompt to compile the query in the foreground. Compiling in the foreground means that you see compiler messages as they scroll onto the screen. It also means that Open M/SQL automatically propmts you to run the query after it has been compiled.

To compile the query in the background, delete the default *No* response, replace it with *Yes*, and press <RETURN>. If you elect to compile the query in the background, you return to the Query Definition template and may continue working while Open M/SQL compiles the query underneath. Open M/SQL does not prompt you to run the query after it has been compiled.

Note InterSystems recommends that you always compile in the foreground.

If you elect to compile the query in the foreground, a series of informational messages will scroll onto the screen as the Compiler moves through its sequence of operations. These messages include a list of the names of the M routines into which the query is being compiled. When the compilation is complete, the Compiler will briefly display the message "...DONE".

After compiling the query (in the foreground), Open M/SQL displays the device selection script to request information about where to send the query output.

You may press the <PREVIOUS> key to back out and not run the query at this time, or you may provide device selection information and proceed with running the query.

For more information on running queries, see the section of this chapter entitled "Running Queries" on page 10-29.

After compiling the query (if you back out before running it) or after running the query, you return to the Query Definition template.

Using the Interactive Query Editor

The Interactive Query Editor provides an alternative, less restrictive way to specify SQL queries.

Unlike the Query Generator, the Interactive Query Editor does not limit you to SELECT-only queries nor does it place you within a prescribed template for query definition. Rather, it provides a free-form SQL editor environment (similar to the Full Screen Editor) where you can create and run any syntactically valid SQL query.

Like the Query Generator, the Query Editor lets you save query definitions for future reuse and save a compiled version of the query to be run at a later time.

Query Types

The table below lists and describes the types of SQL queries supported by Open M/SQL. You may issue queries of any of these statement types via the Interactive Query Editor:

Table 10-7: Query Types Supported By Open M/SQL

Statement Type	Description
ALTER VIEW	Alters the definition of a query-based view. ALTER VIEW queries essentially delete and recreate an existing query-based view, while retaining the same view name and internal view number.
%CHECKPRIV	This query type is an InterSystems SQL extension. Allows a user to check whether or not s/he holds a specified privilege on a specified Open M/SQL object. If the specified privilege exists, the query returns an SQLCODE value of 0 (success). If the privilege does not exist, the query returns an SQLCODE value of 100 (failure). For more information on %CHECKPRIV, see Chapter 9, *Open M/SQL Implementation of SQL*.
CREATE VIEW	Creates a query-based view.
DELETE	Deletes rows from a base table. **Note:** DELETE operates on the entire table, unless a WHERE clause is specified to restrict the scope of the delete.
DROP VIEW	Deletes a query-based view.
GRANT	Grants access privileges on specified objects (base table, view, form, menu object, menu, report, query) to Open M/SQL users with an option to allow the user to grant access to other users.
INSERT	Inserts new rows into a base table.
REVOKE	Revokes access privileges on specified objects (base table, view, form, menu object, menu, report, query) from Open M/SQL users. Only the grantor of privileges has the authority to revoke those privileges.

Table 10-7: Query Types Supported By Open M/SQL (Continued)

Statement Type	Description
SELECT	Retrieves a row or multiple rows from a table and outputs them in standard tabular format. There are two types of SELECT-statement queries: ▫ Non-cursor-based queries (retrieve a single row of data) ▫ Cursor-based queries (retrieve multiple rows of data) The Query Editor automatically converts the syntax of your SELECT statement into a cursor-based SELECT (with all the required SQL DECLARE, OPEN, FETCH, and CLOSE statements). **Note:** You may also use the Query Generator to create cursor-based SELECT statement queries.
UPDATE	Updates column values for one or more existing base table rows. **Note:** UPDATE operates on the entire base table, unless a WHERE clause is specified to restrict the scope of the UPDATE.

Note With the exception of %CHECKPRIV, Open M/SQL implements all of these query statement types according to ANSI-Standard SQL. For information about Open M/SQL-specific extensions and limitations to the syntax for these statements, see Chapter 9, *Open M/SQL Implementation of SQL*.

Creating a Query in the Interactive Query Editor

Using the Interactive Query Editor you may create queries of any statement type supported by Open M/SQL.

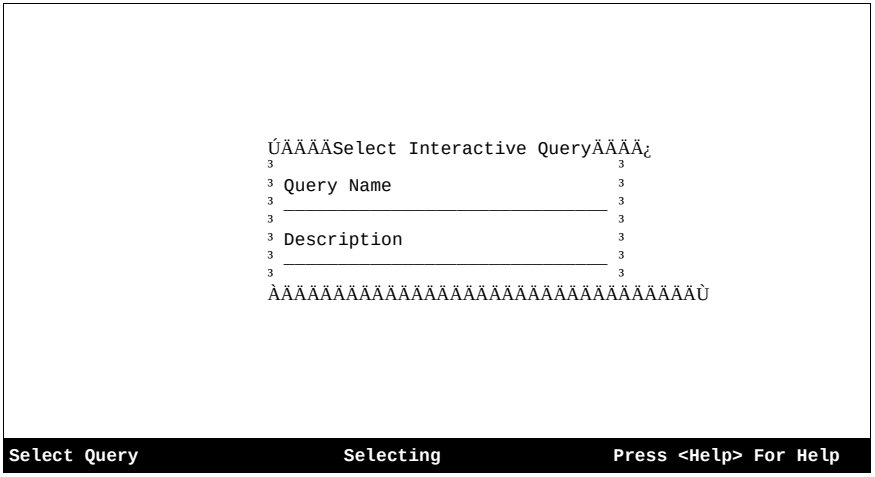
You must give each query you create a query definition name for identification purposes.

Each query definition may consist of only one SQL statement.

Procedure **To create a query using the Interactive Query Editor:**

1. **From the SQL menu, select the Interactive SQL Queries option.**

You see the Query Editor lookup window, as shown below:



2. **At the Query Name field in the Query Editor lookup window, enter a query name, and press <RETURN>.**

3. **At the Description field, you may optionally enter a brief description for the query, and press <RETURN>.**

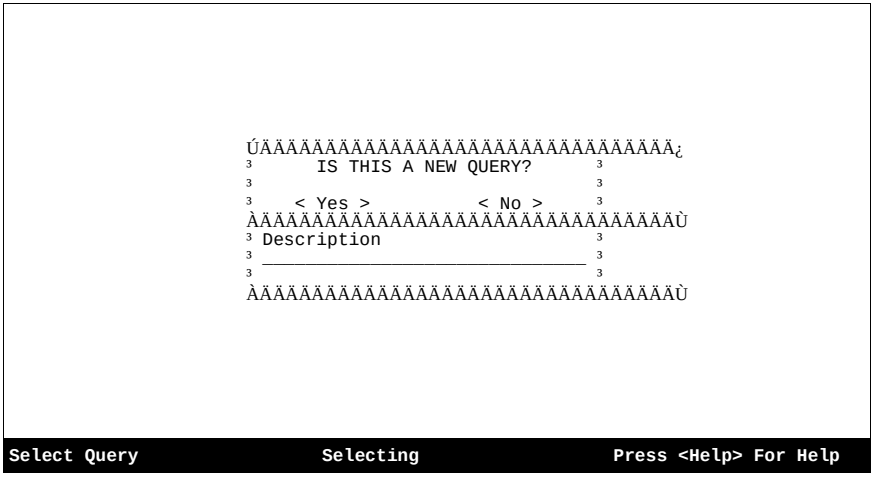
   You can retrieve an existing query to edit or create a new one.

   a. To retrieve an existing query, enter a complete or partial query name in the Query Name field and/or a complete or partial query description in the Description field, and press <RETURN>. You see a lookup box that lists all matching entries.

   To see a lookup box that lists all queries defined in the current database, leave the Query Name and Description fields blank and press the <SEARCH CURRENT TABLE> key. Use the cursor positioning keys to navigate within the lookup box, and press <RETURN> to select an entry.

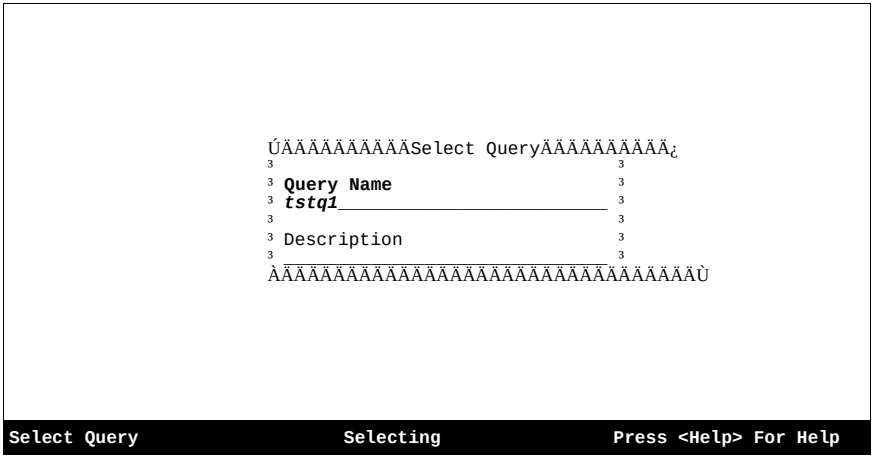
   b. To create a new query, enter a new query name (it must not match the name of any existing query) in the Query Name field, optionally enter a query description in the Description field, and press <RETURN>.

You see the “IS THIS A NEW QUERY?” dialog box, as shown below:



Here, you may create a new query by pressing <RETURN> on the <Yes> action field.

When you have entered a query name (either new or existing), you see the Select Query window, as shown below::



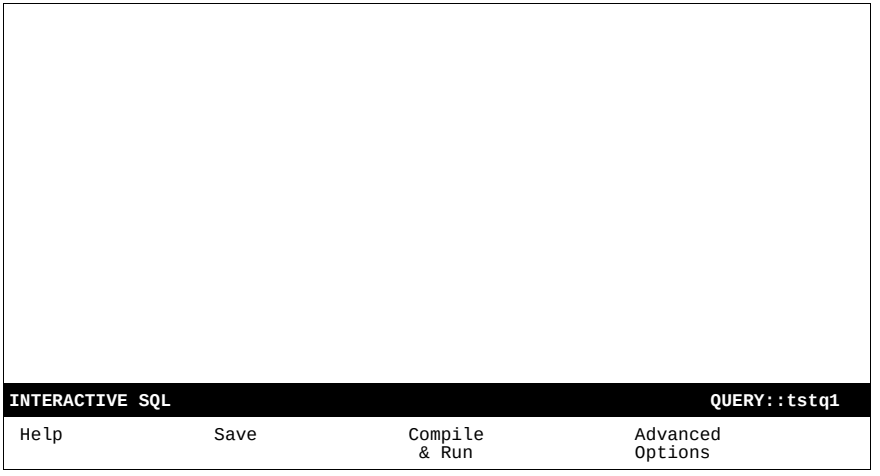
The table below lists and describes the fields located on the Select Query window:

Table 10-8: Fields on Select Query Window

Field	Description
Query Name	This is the name that identifies the query. The name may range from 1 to 30 characters in length and may include any alphanumeric characters, including the underscore character. The name must always begin with an alphabetic character. This is a required field.
Description	Here you may enter a brief description for the query. The description may be up to 60 characters in length and may include any alphanumeric characters, including underscores and blank spaces.

4. Press the <PROCEED> key to invoke the Interactive Query Editor.

The Interactive Query Editor appears as a blank screen, into which you may type your SQL statement.:



At the bottom of the Query Editor Screen is a status line. On the far right, the status line displays the name of the query you are creating.

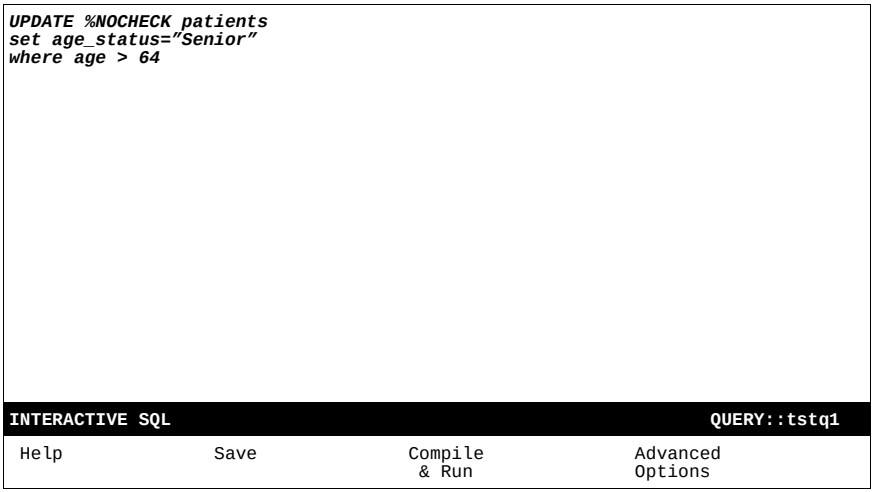
Below the status line is the Query Editor horizontal options menu. See below for more information about the options on this menu.

5. Type in your SQL code without any preprocessor syntax.

SQL commands and identifiers are not case-sensitive, meaning that you may enter them in upper or lower case, or in a mixture of both.

You may enter only one SQL statement per query definition.

The example below shows an UPDATE-statement query called “tstq1” entered via the Query Editor:



Note: You may type the query all on one line, or break it up into any number of lines.

6. **To save your query definition, press the <GO TO BOTTOM MENU> key to access the Query Editor horizontal options menu, and select the Save option from this menu.**

7. **To exit the current query definition, press the <PREVIOUS> key.**

   Be sure to save the query definition before exiting. If there are unsaved changes in your query definition when you attempt to exit, the Query Editor displays the following prompt:

   ```
   Quit Without Saving Changes?
   ```

   Type *Y* (to quit) or *N* (to return to the Editor), and press <RETURN>.

   When you exit the Query Editor, you return to the Query Editor lookup window, where you may create or retrieve another query definition.

8. **From the Query Editor lookup window, press the <PREVIOUS> key again to return to the SQL menu.**

Editing Commands

The table below lists and describes the editing capabilities supported by the Interactive Query Editor:

Table 10-9: Editing Capabilities Supported by the Interactive Query Editor

Function	Description	Keystrokes to Use
Select	Selects a block of text for cutting and pasting.	Press the <LIST CHOICES> key, or <CTRL-E> <S>
Cut	Deletes a block of selected text.	Press the <REMOVE> key, or <CTRL-E> <C>
Paste	Pastes a block of selected text at the point where the cursor is located.	Press the <INSERT> key, or <CTRL-E> <C>

Query Editor Horizontal Options Menu

Located at the bottom of the Query Editor display screen is the Query Editor horizontal options menu, which contains options that provide additional functionality to the Query Editor environment.

To access the Query Editor horizontal options menu, press the <GO TO BOTTOM MENU KEY> key from within the Query Editor.

To select an option from the Query Editor horizontal options menu, use the <RIGHT ARROW> and <LEFT ARROW> keys to position the select bar on a desired option and press <RETURN>, or type the first letter in the name of a desired option to position the select bar, and press <RETURN>.

The table below lists and describes the options on the Query Editor horizontal options menu:

Table 10-10: Options on Query Editor Horizontal Options Menu

Options	Description
Help	Select this option to invoke a submenu of on-line help options
Save	Select this option to save your query definition in its current form. After saving, this option returns you to the Query Editor to continue working.
Compile & Run	Select this option to compile and run the query.
Advanced Options	Select this option to access the Query Definition Advanced Features window, which displays various information about the query and allows you to define parameters for printing the query in ASCII-delimited output format. **Note:** Output formatting is applicable to SELECT-statement queries only.

Using On-line Help

The Interactive Query Editor provides a Help submenu for accessing on-line help on any of the following topics:

- General Help
- SQL Syntax
- Select
- Insert
- Update
- Delete
- Grant & Revoke
- %CHECKPRIV

Procedure **To access the Help submenu:**

1. **From the Query Editor screen, press the <GO TO BOTTOM MENU> key.**

   The Help submenu replaces the top-level Query Editor horizontal options menu at the bottom of the screen, as shown below:

2. **To select an option from the Help submenu, use the <RIGHT ARROW> and <LEFT ARROW> keys to position the select bar on a desired option and press <RETURN>, or type the first letter in the name of a desired option to position the select bar, and press <RETURN>.**

   This displays a series of on-line help screens that provide detailed information on the selected topic.

3. **To exit an on-line help screen, press the <PREVIOUS> key.**

Query Editor Advanced Options

You may use the Advanced Options option on the Query Editor horizontal options menu to access the Query Editor Advanced Options window.

The Query Editor Advanced Options window displays the following information about your query:

- Owner (this value is read-only)
- Run-Time Measure (this value is read-only)
- Routine Name (this value is modifiable)

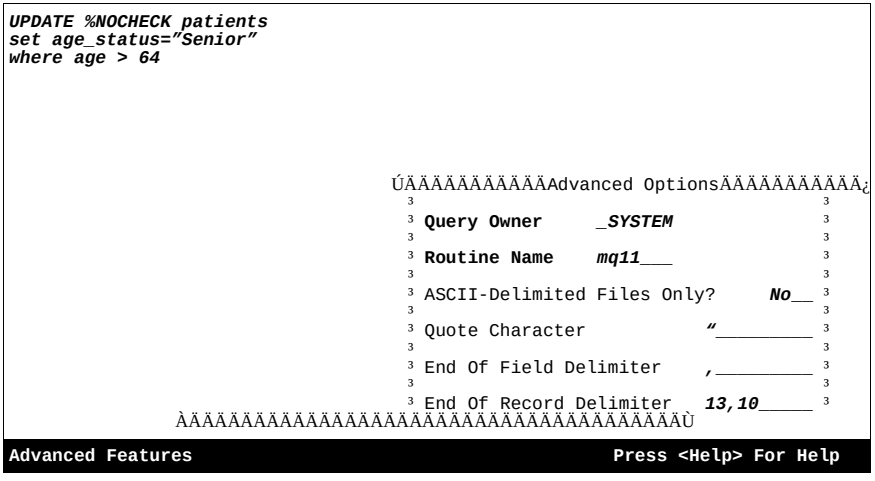
The Query Editor Advanced Options window also lets you define parameters for printing your query in ASCII-delimited output format.

Open M/SQL supports the output of SELECT-statement queries to ASCII-Delimited text files, which are formatted for import into many PC software packages.

Procedure **To access and use the Query Editor Advanced Options window:**

1. **From the Query Editor screen, press the <GO TO BOTTOM MENU> key to access the Query Editor horizontal options menu.**
2. **From the Query Editor horizontal options menu, select the Advanced Options option.**

   You see the Query Editor Advanced Options window, as shown below.



The following table lists and describes the fields located on the Query Editor Advanced Options window:

Table 10-11: Fields on Query Editor Advanced Options Window

Field	Description
Query Owner	This field shows the UserName of the user who created (and therefore owns) the query. You cannot edit this field.
Run Time Measure	This field provides a relative measure of the system cost associated with running the query. The number shown is based on the complexity of both the query and the database. Typically, the larger the number, the more slowly the query executes. Open M/SQL calculates the run-time measure when you compile the query and recalculates it every time you recompile. This value is read-only and cannot be altered in any way.
Routine Name	This field displays the prefix for all routine names generated by the query. Open M/SQL assigns a default prefix, such as “mq11” in the above example. You may optionally override the default prefix by replacing it with a string of 1 to 7 alphanumeric characters (first character must be alphabetic) to serve as the override prefix. All routines generated for the query consist of this prefix followed by a single character (1-9, then A-Z, then a-z), which is appended by the compiler. Specifying your own routine prefix may help you to remember it when attempting to call the query directly from M using the query call syntax, as described in the section of this chapter entitled “Running a Query from M Code” on page 10-31.
ASCII-Delimited Files Only?	Here you may specify ASCII-Delimited as the only allowable output format for the query, i.e. no printed format is allowed. Answer *Yes* to specify ASCII-Delimited as the only allowable output format for the query. This enables you to download an unlimited number of fields to the ASCII-Delimited file. Answer *No* to not restrict output of the query to ASCII-delimited format only. If you answer *No*, you can still send query output to an ASCII-Delimited file, but the number of fields you may select is limited to the number of fields supported by a standard printed query. *No* is the default response. **Note:** Output formatting is applicable to SELECT-statement queries only. For more information on sending query output to ASCII-delimited files, see the section of this chapter entitled “ASCII-Delimited Output for Queries” on page 10-36.
Quote Character	Here you may specify the character(s) used to enclose field values when using ASCII-delimited output format. You may specify any text of ten characters or less. The default is the double quote character (“).

Table 10-11: Fields on Query Editor Advanced Options Window (Continued)

Field	Description
End of Field Delimiter	Here you may specify the character(s) used to act as the separator between field values when using ASCII-delimited output format. You may specify any text of ten characters or less. The default is the comma character (,).
End of Record Delimiter	Here you may specify the character(s) used to act as the separator between records (rows) when using ASCII-delimited output format. You may specify a list of ASCII values (numbers) separated by commas. The default value is the ASCII sequence 13,10 (same as <CR><LF>).

3. **At the Routine Name field, you may optionally replace the default routine prefix assigned to your query by Open M/SQL with an override routine prefix.**

   The routine prefix may be 1 to 7 characters in length and may include any alphanumeric characters, except for the first character, which must be alphabetic.

4. **At the ASCII-Delimited Files Only? field, answer *Yes* or *No* to indicate whether or not you want to restrict the query to ASCII-Delimited output format only.**

   No is the default response.

5. **At the Quote Character field, specify the character(s) used to enclose field values when using ASCII-delimited output format.**

   The default is the double quote character (“).

6. **At the End of Field Delimiter field, specify the character(s) used to act as the separator between field values when using ASCII-delimited output format.**

   The default is the comma character (,).

7. **At the End of Record Delimiter Field, specify the character(s) used to act as the separator between records (rows) when using ASCII-delimited output format.**

   The default value is the ASCII sequence 13,10 (same as <CR><LF>).

8. **When you complete your work in the Query Editor Advanced Options window, press the <PROCEED> key to save your definitions and exit.**

Compile and Run the Query

You may use the Compile & Run option on the Query Definition menu bar to compile and run your query.

When you select this option, Open M/SQL automatically (re)compiles the query.

As the Compiler moves through its sequence of operations, informational messages will scroll onto the screen. These messages include a list of the names of the M routines into which the query is being compiled. When the compilation is complete, the Compiler will briefly display the message "...DONE".

After compiling the query, Open M/SQL runs it.

INSERT, UPDATE, DELETE, GRANT, REVOKE, CREATE VIEW, ALTER VIEW, and DROP VIEW statement queries produce no examinable output. When you run a query of any of these types, Open M/SQL returns a success message (if the query has completed successfully) or an error message (if the query has failed) to your screen, for example:

```
'tstq1' Query Successfully Completed
```

%CHECKPRIV queries return a message to the screen that indicates the existence/nonexistence of the queried privilege, for example:

```
The Requested Privilege Exists
```

SELECT-statement queries generate examinable output that must be directed to a specified destination. Open M/SQL displays a device selection script to request information about where to send the query output.

You may press the <PREVIOUS> key to back out and not run the query at this time, or you may provide device selection information and proceed with running the query.

For more information on running SELECT-statement queries (output device selection and output formatting), see the section of this chapter entitled "Running Queries" on page 10-29.

After compiling the query (if you back out before running it) or after running the query, you return to the Query Editor screen.

Running Queries

You can run a query from any of three locations:

- From within its definition environment (Query Generator or Query Editor)
- Using the Run Existing Queries utility on the SQL menu
- From M code

Privileges Required to Run Queries

In order to run any query, you must hold SELECT privileges on the query definition.

Furthermore, in order to run certain query types you must hold certain privileges on the base tables named in the query, as described in the table below:

Table 10-12: Privileges Required to Run Query Types

Query Type	Privileges Required
SELECT	Must have SELECT privileges on each of the base tables named in the query.
INSERT	Must have INSERT privileges on each of the base tables named in the query.
UPDATE	Must have %ALTER privileges on each of the base tables named in the query.
DELETE	Must have DELETE privileges on each of the base tables named in the query.

Running a Query From its Definition Environment

You can run queries directly from the Query Generator or the Query Editor.

Procedure **To run a query from within the Query Generator:**

1. **From the Query Definition template, press the <ENHANCE> key twice to access the Query Definition menu bar.**
2. **From the Query Definition menu bar, type *c* to select the Compile & Run option.**

   Open M/SQL automatically (re)compiles the query, and then displays a device selection script to request information about where to send the query output (see below for a discussion of the Device Selection window). Remember that all queries defined using the Query Generator are SELECT-statement queries, and therefore produce examinable output.

Procedure **To run a query from within the Query Editor:**

1. **From the Query Editor screen, press the <GO TO BOTTOM MENU> key to access the horizontal options menu.**

2. **Select the Compile & Run option from the Horizontal Options menu and press <RETURN>.**

   Open M/SQL automatically (re)compiles the query and then runs it.

   If it is an UPDATE, INSERT, DELETE, GRANT, REVOKE, %CHECK-PRIV, CREATE VIEW, ALTER VIEW, or DROP VIEW statement query, the query returns a success/fail message to the screen when it finishes.

   If it is a SELECT statement query, Open M/SQL displays a device selection script to request information about where to send the query output (see below for a discussion of the Device Selection window).

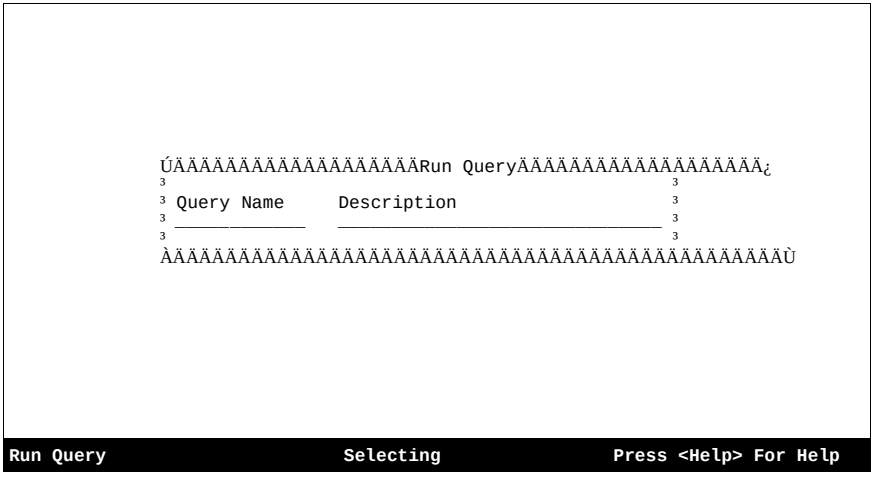
Using the Run Existing Queries Utility

You can use the Run Existing Queries utility to select any query defined via the Query Generator or Query Editor and run it.

Procedure **To use the Run Existing Queries utility:**

1. **From the SQL menu, select the Run Existing Queries option.**

   You see the Run Query lookup window, as shown below:



2. **To select a query, enter a complete or partial query name in the Query Name field and/or a complete or partial query description in the Description field, and press <RETURN>.**

You see a lookup box that lists all matching entries.

To see a lookup box that lists all queries defined in the current database, leave the Query Name and Description fields blank and press the <SEARCH CURRENT TABLE> key.

3. **Use the <UP ARROW> and <DOWN ARROW> keys to navigate within the lookup box, and press <RETURN> to select an entry.**

4. **When your have selected a query, press the <PROCEED> key to execute it.**

   If the query has not been recompiled since changes were last saved, Open M/SQL will automatically recompile the query before executing it.

Running a Query from M Code

It is also possible to run a query by calling it directly from M code.

To do this, you use the M query call syntax. You may use this entry point to invoke a query in any of the following ways:

- From a trigger item of action type M Code
- From menu option of action type M code
- From within an M routine
- From the M programmer prompt

The M query call syntax is as follows:

```
do query^%msql(queryname,user,device,format,batchsw)
```

The following table lists and describes the parameters associated with the M query call syntax:

Table 10-13: Parameters Associated with M Query Call Syntax

Parameter	Meaning
queryname	The name of the query you want to run. This is a required parameter.
user	This is a moot parameter; Open M/SQL no longer uses this information. Enter the null indicator ("") to skip this parameter.
device	The optional name of an output device.

Table 10-13: Parameters Associated with M Query Call Syntax (Continued)

Parameter	Meaning
format	The optional name of a device format.
batchsw	An optional flag that specifies whether to run the query in the foreground or in the background. Set to 1 display the query results in the background. Set to 0 to display the results in the foreground. 0 is the default setting.

Selecting an Output Device

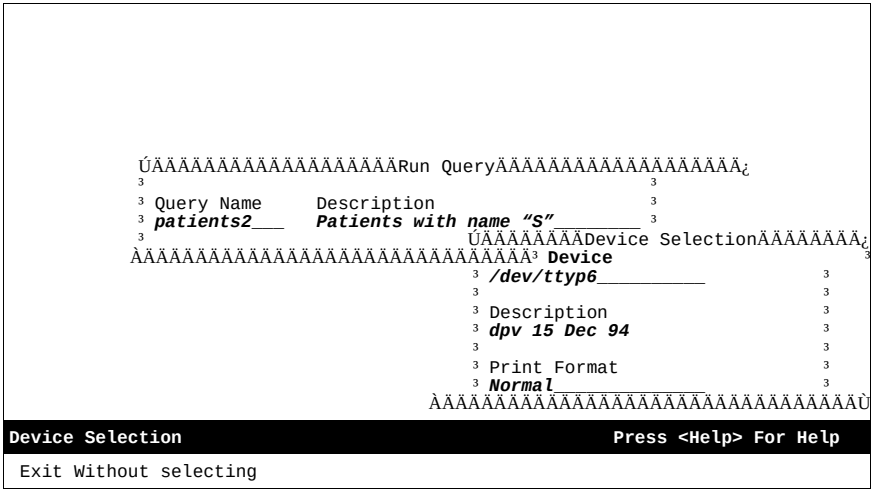
When you run a SELECT-statement query, Open M/SQL invokes a device selection script to request information about where to send the query output. You must specify an output device as well as print format parameters for the device.

The appearance of device selection script differs across the various M systems supported by Open M/SQL. When you run Open M/SQL on top of a DSM, DTM, or Micronetics' MSM host M system, Open M/SQL calls out to the device handling routine supported by your M system. Refer to your implementation-specific M programming manual for information on how your device handling routine works.

Procedure **To select an output device for a SELECT-statement query:**

1. **Run a SELECT-statement query.**

   If you are running Open M/SQL on an ISM system, you see the Device Selection window, as shown below:



The following table lists and describes the fields located on the Open M/SQL for ISM Device Selection window:

Table 10-14: Fields on Open M/SQL for ISM Device Selection Window

Field	Description
Device	This field always defaults to your current device (the current value of $IO). You may change the default to any valid output device to which your device is linked.
Description	This field reflects the description given to the specified device in the device table.
Print Format	Here you may select any print format defined for the specified device.

2. **In the Device field, enter the name of the device to which you want to send the query.**

   You can send the query to any valid output device that is linked to your current device.

   The default device is your current device.

   To send the query to your screen, press the <PROCEED> key.

   To send the query to another device (such as a printer), delete the name of the default device at the Device prompt, enter the name of the new target device, adjust the Print Format parameter as appropriate, and press the <PROCEED> key.

3. **When you have entered a device name and the appropriate print format parameters, press the <PROCEED> key.**

   This executes the query.

The example below shows sample output for the following query:

```
SELECT name,sex,age,AVG(age %FOREACH(sex))
   FROM Patients
   WHERE name %STARTSWITH "S"
   ORDER BY name
```

```
Name                      Sex     Age   expression
_________________________ ______  ____  __________
Salisbury, Harvey         Male    51    49
Sanderson, George         Male    63    49
Sawyer, Eleanor           Female  77    57
Schmidt, Mitchell         Male    44    49
Scott, Denise             Female  46    57
Scott, Michael            Male    33    49
Shapiro, Oscar            Male    79    57
Silva, Louise             Female  61    57
Simmons, Virginia         Female  84    57
Smith, Charlotte          Female  91    57
Smith, Joe                Male    49    49
Smith, Timothy            Male    19    49
Snyder, Estelle           Female  67    57
Spring, Jonathon          Male    79    49
Stevens, Charles          Male    60    49
Stevens, Theresa          Female  22    57
Stewart, Frederick        Male    37    49
Stone, Julia              Female  31    57
Sullivan, Betty           Female  42    57
Sullivan, Wayne           Male    46    49
Sweeney, Terrence         Male    28    49
Press <Return> to continue, <Options> to scroll, <Exit> to Exit
```

Note: When output to the screen, the query displays one screen at a time and prompts you to press <RETURN> to scroll ahead to the next screen.

Device Selection for ASCII-Delimited Output Format

To print a query in ASCII-Delimited output format, enter "ASCII-Delimited" at the Print Format field.

If you designated the query for ASCII-Delimited output only (this option is located in the Advanced Options window), the Print Format field in the Device Selection window automatically defaults to "ASCII-Delimited" and cannot be changed.

Note that Open M/SQL does not require that you specify a file as the output device in order to generate ASCII-Delimited output. You may find it useful to send ASCII-Delimited records to a terminal device, such as an asynchronous communications line, or to a spooler device.

ASCII-Delimited Output for Queries

Open M/SQL supports the ability to direct the output from SELECT-statement queries defined via the Query Generator or Interactive Query Editor to ASCII-Delimited text files. These files are formatted for import into many PC software packages.

Contents of an ASCII-Delimited File

In an ASCII-Delimited file, each record consists of one or more field(s) enclosed in a user-defined quotation character and delimited by a user-defined delimiter character. A user-defined end-of-record sequence follows each record. A sample record might appear as follows:

```
"3","Smith,John","12345","Male","675.2","","","Fractured
Tibia"<CR><LF>
```

where the Quote Character is ("), the End of Field Delimiter is (,), and the End of Record Delimiter is (<CR><LF>).

All occurrences of the Quote Character will be stripped from your data prior to the formatting of output records.

For each detail line output, Open M/SQL generates a sequential record identifier, which may be used as a key field to identify the record within your application. The first field of each record will always be the record identifier. All other fields are output in the order they were specified in the query definition. If a SELECT statement query was used, the output columns are ordered by table column number.

Open M/SQL does not restrict the maximum length for ASCII-Delimited output records. However, you should be aware of any record length limitations imposed by the application importing the ASCII-Delimited file.

Queries Support Dual Output Formats

In Open M/SQL, a single query definition can produce both standard printed output and ASCII-Delimited file output.

For standard printed queries, Open M/SQL currently restricts the maximum number of fields that can you can select. To surmount this limitation you may designate the query for ASCII-Delimited output only. By choosing this option you relinquish all formatting control over the output, but gain the privilege of including an unlimited number of fields in the ASCII-Delimited output file. This choice is also reversible, i.e., you may initially designate a query for ASCII-Delimited output only and later modify it to allow both types of output. However, if you do this you may need to reduce the number of query fields in order to permit printed output.

Copying Queries

You may use the Copy Query utility to make a copy of a query definition.

This utility creates a copied query that is virtually identical to the source query.

As soon as the copy is made, the copied query relinquishes all ties to its source query. You may use the copied query in all the same ways you use a regular query. Specifically, you may:

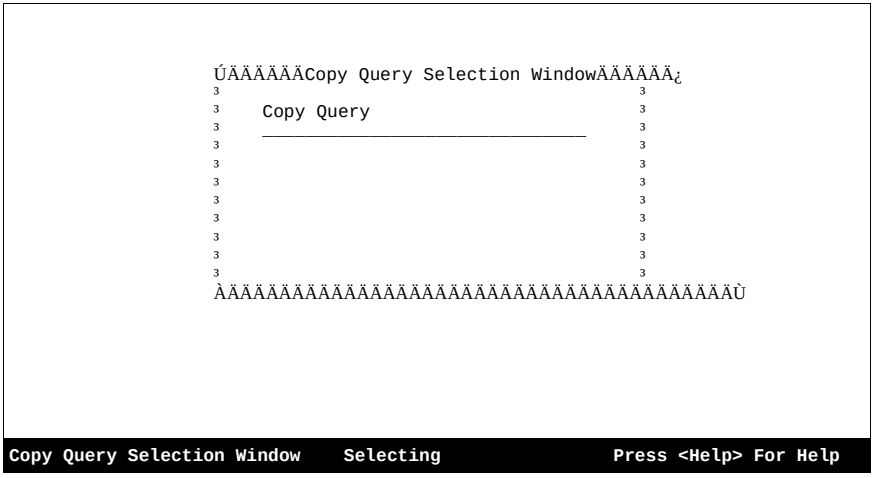
- Edit it
- Run it
- Use it as the data source of a view or report
- Use it in the FROM clause of another query

You may copy any query, regardless of whether or not you hold privileges on it. However, in order to run a copied query, you must hold SELECT privileges on the source query. And in order to edit a copied query, you must hold %ALTER privileges on the source query.

Procedure **To copy a query:**

1. **From the SQL menu, select the Copy Query option.**

   You see the Copy Query Selection lookup window, as shown below:



2. **At the Copy Query field, enter the name of the source query (the query you want to copy), and press <RETURN>.**

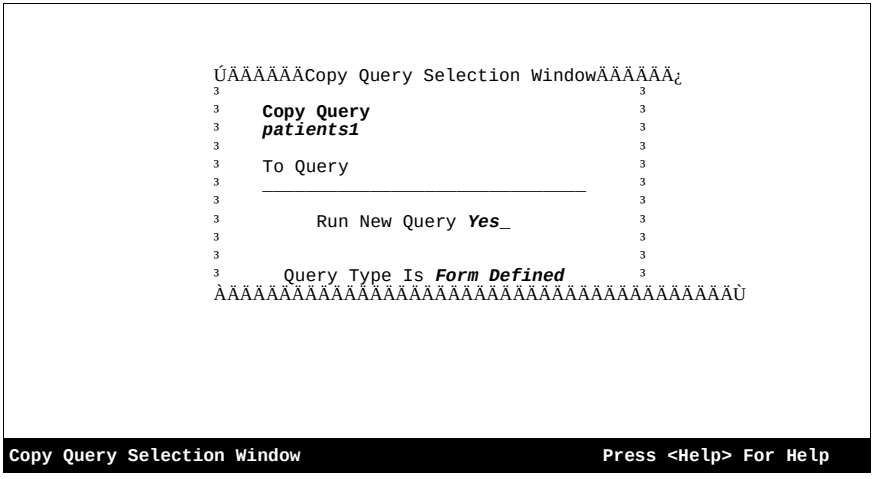
You may select any query defined via the Query Generator or the Query Editor.

To select a query, enter the complete or partial name of an existing query, and press <RETURN>. You see a lookup box that lists all matching entries.

To see a lookup box that lists all queries defined in the current database, leave the Copy Query field blank and press the <SEARCH CURRENT TABLE> key.

3. **Use the cursor positioning keys to navigate within the lookup box, and press <RETURN> to select an entry.**

The Copy Query Selection window fills out with its complete set of fields, as shown below:



4. **At the To Query field, enter the name of the new query to be created.**

A query name may range from 1 to 30 characters in length and may include any alphanumeric characters, including the underscore character. The name must always begin with an alphabetic character.

This name must not already be in use by another query.

5. **At the Run New Query? field, answer *Yes* or *No* to indicate whether or not you want to compile and run the new query as soon as it is copied.**

Answer Yes to compile and run the new query immediately after creating it.

No is the default response.

The Query Type Is field shows the type of query that you have specified as the source query. This field may have either of the following two values:

- Interactive (defined in the Query Editor)
- Form-Defined (defined using the Query Generator)

6. **Press the <PROCEED> key to copy the query.**

   When the copy is complete, Open M/SQL displays the following message at the bottom of the screen:

   ```
   Query Copy Successful
   ```

   If you typed *No* in the Run New Query? field, you return to the SQL menu.

   If you typed *Yes* in the Run New Query? field to request Open M/SQL to compile and run the query immediately after creating it, you see the following message at the bottom of the screen:

   ```
   Compiling and running Query <query name>
   ```

   After compiling the query, Open M/SQL displays the appropriate device selection script for your M system to request information about where to send the query output.

7. **Enter a device name and appropriate print format parameters, then press the <PROCEED> key to execute the query.**

List Queries Report

The List Queries utility lets you generate a screen or hard copy report that lists all queries defined in the current database (via the Query Generator and Interactive Query Editor) and provides the following information about each one.

- **Query Name** — Name of the query (queries are listed in alphabetical order)
- **Query Description** — Description given to the query

Procedure **To run the List Queries report:**

1. **From the SQL menu, select the List Queries option.**

   You see the device selection script native to your host M system.

   If you are running Open M/SQL on an ISM system, you see the Device Selection window.

2. **Enter the name of the output device to which you want to send this report and specify the appropriate print format parameters.**

3. **When you have entered a device name and the appropriate print format parameters, press the <PROCEED> key.**

   This executes the report.

   Below is a sample List Queries report:

```
Query                          Query
Name                           Description
-----------------------------  -----------------------------
ages
Census1994                     Census for year 1994
CensusMay95                    Count Patients for May 95
Count_Admissions               Total Number of Admissions
CountPatients                  Count of Number of Patients
ListMRN                        List Current Medical Rec. #s
Males65                        List Males over 65
Occupancy                      Calculate Occupancy
oldpats
Patient_Count                  Count of Patient Entries
patients1                      Average Age by Ward, Sex
patients2                      Patients with name "S"
PatStates
personnel
Revenue1994                    Revenue for Year 1994
RevenueMay94                   Revenue for May 1994
States
tstq1                          Test Query
Press <Return> to Exit
```

   Note: When output to the screen, this report displays one screen at a time and prompts you to press <RETURN> to scroll ahead to the next screen.

4. **Press the <PREVIOUS> key to exit the List Queries report and return to the SQL menu.**

Detailed Query Listing Report

The Detailed Query Listing utility lets you generate a screen or hard copy report that lists a specific query, all queries, or a range of queries whose query names begin with a similar character string.

The Detailed Query Listing Report provides the following information about each query:

- **Query Name** — Name of the query (queries are listed in alphabetical order)
- **Query Description** — Description given to the query
- **Query Owner —** UserName of the user who created the query
- **Routine Prefix** — Name of the routine prefix used by all routines generated for the query by Open M/SQL
- **SQL Text** — Text of the query

All queries defined in the current database (via the Query Generator and Interactive Query Editor) are available to this report.

Procedure **To run the Detailed Query Listing report:**

1. **From the SQL menu, select the Detailed Query Listing option.**

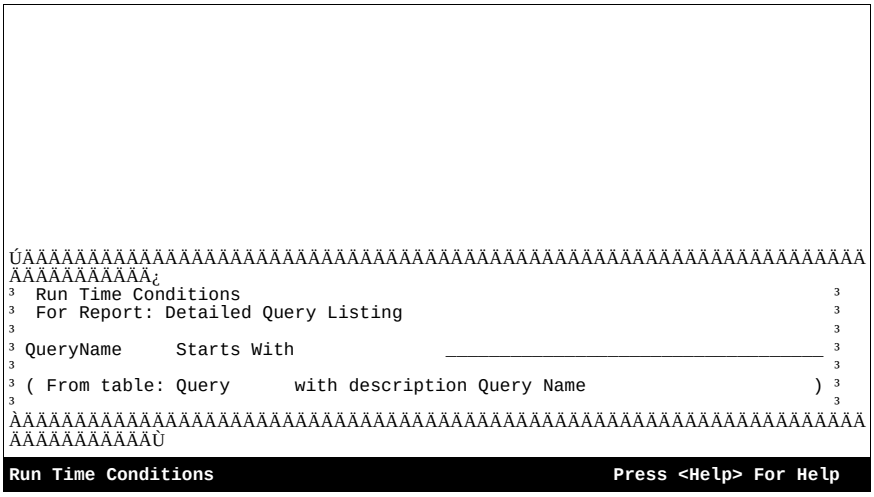
   You see the device selection script native to your host M system.

   If you are running Open M/SQL on an ISM system, you see the Device Selection window.

2. **Enter the name of the output device to which you want to send this report and specify the appropriate print format parameters.**

3. **When you have entered a device name and the appropriate print format parameters, press the <PROCEED> key.**

You see the Detailed Query Listing Run-Time Conditions window, as shown below:



4. **In the Starts With field specify the query or queries you want to list in the report.**

   To specify all queries defined in the current database, leave the Starts With field blank.

   To specify just the “patients1” query, type the literal string “patients1”.

   To specify all queries that begin with the letter “P”, type the letter “P”.

5. **Press the <PROCEED> key to execute the report.**

Below is a sample Detailed Query Listing report for all queries that begin with the letter “P”:

```
                                                         Date: 17 Jul 95  Page: 1

                            InterSystems Corporation
                             Detailed Query Listing
                  ============================================


Query: patients1               Description: Average Age By Ward, Sex
Owner: _SYSTEM                 Routine:     mq9

SQL:  SELECT ward,sex,AVG(age)
      FROM Patients
      GROUP BY ward,sex
      ORDER BY ward,sex
      HAVING AVG(age) < 62

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=


Query: patients2               Description: Patients with name “S”
Owner: _SYSTEM                 Routine:     mq10

SQL:  SELECT name,sex,age,AVG(age %FOREACH(sex))
      FROM Patients
      WHERE Name %STARTSWITH “S”
      ORDER BY name

Press <Return> to continue, <Options> to scroll, <Exit> to Exit
```

Note: When output to the screen, this report displays one screen at a time and prompts you to press <RETURN> to scroll ahead to the next screen.

6. **Press the <PREVIOUS> key to exit the Detailed Query Listing report and return to the SQL menu.**

PART

V

Application Programming

Chapter 11

Programmer Interface to Applications

Chapter 12

Open M/SQL Developer Utilities

CHAPTER

11

Programmer Interface to Applications

In Open M/SQL, you can insert M and SQL code at designated locations within object definition windows to customize forms, reports, and the underlying base table operations of lookup and filing to your precise specifications. Specifically, this inserted code can customize base table and window triggers, lookup queries, filing routines, and operations that calculate, validate, and convert data fields. In each case, the inserted code must observe specific conventions for communicating with the application that calls it.

Open M/SQL supplies entry points that enable you to call forms, reports, queries, and menus from routines as well as from the M programmer prompt. Other utility entry points perform functions that emulate forms. All entry points observe specific conventions for parameter passing.

Embedded SQL provides a way for an M program to operate the machinery of retrieval and filing within Open M/SQL base tables.

This chapter discusses the following topics:

- Open M/SQL Variables
- Entry Points to the %msql Routine
- Open M/SQL Globals
- Inserting Code into Open M/SQL Applications
- Calling Open M/SQL Objects from M Programs
- Emulating Form Behavior from M Programs
- Establishing Authorization ID from Programmer Mode
- Intermixing Open M/SQL Objects with User-Defined M Routines
- Contents of Applications

Open M/SQL Variables

All local variables used by Open M/SQL begin with the percent sign (%) and have all-lowercase names. A list of these variable names appears below. Do not use these names for any other purpose than what is documented. InterSystems reserves the right to add to this list of percent variables in future releases, and therefore strongly advises that you do not use any user-defined variables that begin with the percent sign.

Open M/SQL uses variables with lowercase names to run forms and reports. To avoid the possibility of conflict, InterSystems recommends that you do not use all-lowercase variable names in any M code you insert into your Open M/SQL applications or in any routines called by your applications.

Embedded SQL uses its own percent variables without protecting them by a NEW command. These variables all begin with the prefix %mmm and thus do not run a high risk of conflict. When defining variables in a program containing embedded SQL, make sure your variables do not begin with %mmm.

List of Open M/SQL Percent (%) Variables

The following table lists and describes the percent variables used by Open M/SQL to communicate with applications, or used by programs that call applications.

You will note that some of these percent variables are enclosed in curly braces, such as {%action}. The curly brace syntax indicates that these are not raw M variables, but rather Open M/SQL interpreted variables. Variables of this syntax remain the same even when the names of the underlying variables they represent change.

Table 11-1: Open M/SQL Percent (%) Variables

%variable	Function	Where Used
%msql	Stores the current UserName and Terminal Type, as well as other user preferences.	General
%is() array	Stores terminal-specific command strings to be executed by routines.	General
%mode() array	Stores terminal-specific appearance attributes to be written by routines.	General
%val	Stores the data value of a field for use in external-to-internal and internal-to-external conversion and user-defined additional validation code.	Data Dictionary & Form Generator
%ok	Flag that can be set to 0 to disallow the filing of an invalid field value.	Data Dictionary & Form Generator

Table 11-1: Open M/SQL Percent (%) Variables (Continued)

%variable	Function	Where Used
%msg	Stores text message returned by validation code when rejecting an invalid field value.	Data Dictionary & Form Generator
%data(icol)	Stores the data value of the current field of the current row, whether saved or unsaved.	Form Generator
%edit(icol)	Stores the disk value of the current field of the current row when that field has been edited but not saved. Unless %data(icol) is unsaved, %edit is undefined.	Form Generator (for database fields only)
%new	Stores the new value of an edited field. This variable is used to allow validation code to examine the new field value.	Form Generator
%newext	Stores the new external value of an edited field. This variable is used to allow validation code to examine this value.	Form Generator
%old	Stores the old value of an edited field. This variable is used to allow validation code to examine the old field value.	Form Generator
%oldext	Stores the old external value of an edited field. This variable is used to allow validation code to examine this value.	Form Generator
{%parent_reference}	For forms based on child tables, {%parent_reference} holds the parent reference ID (the RowID of the current row in the parent table). For forms not based on child tables, the value of {%parent_reference} is "".	Form Generator Specifically: n All form trigger locations n Variable Window Placement n Form-only Field Compute Code
{%caller}	Stores the current value of the Caller ID.	Form Generator Specifically: n All form trigger locations n Variable Window Placement n Menu Object Definition

Table 11-1: Open M/SQL Percent (%) Variables (Continued)

%variable	Function	Where Used
{%action}	Stores the last keyboard action entered by the user. It can have any of the following values: n PROCEED n PREVIOUS n GETOUT n GETOUTALL n Up Arrow n Down Arrow n Return n Tab	Form Generator Specifically: n Post-Form triggers n Post-Field triggers n Post-Window triggers
{%filetype}	Stores the type of filing currently in effect. Possible values are: n "Update" — indicates that the current form is in data entry mode. n "Inquiry" — indicates inquiry mode. n "Insert"— indicates that you have just confirmed insertion of a row.	Form Generator Specifically: n Post-Form triggers n Post-Window triggers n Post-Field triggers n Pre-Row triggers n Post-Row triggers n Variable Window Placement
{%presave}	Flag that indicates whether or not data ha been saved within the current form. If %savedata is 1, data has been saved. If it is 0, no data has been saved.	Form Generator Specifically: n Post-Form triggers n Post-Window triggers n Post-Field triggers n Variable Window Placement
{%savedata}	Flag that indicates whether or not the current form contains unsaved data. If %savedata is 1, there is unsaved data within the current form. If it is 0, there is no unsaved data.	Form Generator Specifically: n Post-Form triggers n Post-Window triggers n Post-Field triggers n Variable Window Placement
{%timeout}	Flag that indicates whether or not the form was exited as the result of a time-out action. If %timeout is 1, the form did time out, If it is 0, the form did not time out.	Form Generator Specifically: n All form trigger locations

Table 11-1: Open M/SQL Percent (%) Variables (Continued)

%variable	Function	Where Used
{%return_action}	Stores the value of %action from the form that was last exited. It can have any of the following values: n PROCEED n PREVIOUS n GETOUT n GETOUTALL n Up Arrow n Down Arrow n Return n Tab	Form Generator Specifically: n All form trigger locations
{%return_filetype}	Stores the value of %filetype from the form that was last exited.	Form Generator Specifically: n All form trigger locations
{%return_presave}	Stores the value of %presave from the form that was last exited.	Form Generator Specifically: n All form trigger locations
{%return_savedata}	Stores the value of %savedata from the form that was last exited.	Form Generator Specifically: n All form trigger locations except Pre-Row triggers
{%return_timeout}	Stores the value of %timeout from the form that was last exited.	Form Generator Specifically: n All form trigger locations
{%inquiry_mode}	Flag that indicates whether or not the form is currently in inquiry-only mode. I %inquiry is 1, the form is in inquiry-only mode. I it is 0, the form is not in inquiry-only mode.	Form Generator Specifically: n All form trigger locations n Menu Object Definition
{%linenum}	Stores the line number of the current line in a multi-line field.	Form Generator Specifically: n Field Definition for multi-line fields

Table 11-1: Open M/SQL Percent (%) Variables (Continued)

%variable	Function	Where Used
{%retrieved}	Flag that is set to 1 if any rows are retrieved when a multi-row form is run, 0 otherwise. This variable applies to multi-row forms only.	Form Generator Specifically: ▪ All form trigger locations except Pre-Form and Pre-Window triggers
{%menubar}	Stores the name of the currently active menu bar.	Form Generator Specifically: ▪ All form trigger locations except Pre-Row and Post-Row triggers ▪ Menu Object Definition
{%menuid}	Stores the internal identification number of currently active menu bar.	Form Generator Specifically: ▪ All form trigger locations except Pre-Row and Post-Row triggers ▪ Menu Object Definition
{%menutype}	Stores the type of the currently active menu object. Its value is either “menu bar” or “pop-up menu”.	Form Generator Specifically: ▪ Menu Object Definition
{%cellar_tuple}	Flag that is set to 1 when the user of a multi-row form inserts a row at the bottom of the form.	Form Generator Specifically: ▪ Pre-Row triggers
{%date}	Stores the current date in an M/PACT report.	M/PACT
{%time}	Stores the current time in an M/PACT report.	M/PACT
{%agg}	Stores the calculation for the aggregate function you have specified.	M/PACT
{%pagenum}	Stores the current page number in an M/PACT report.	M/PACT
{%newpage}	Forces a page feed in an M/PACT report.	M/PACT

Table 11-1: Open M/SQL Percent (%) Variables (Continued)

%variable	Function	Where Used
{%report-end}	Flag to indicate whether or not a report ran to completion. If user exits report before completion by pressing the <PREVIOUS> key, {%report-end} is set to “user abort”. If the report ran to completion, it is set to NULL (“”).	M/PACT Specifically: n Post-Report triggers
{%background}	Flag to indicate whether or not a report is being run in the foreground. If the report is being run in the foreground, {%background} is set to 0. If a report is being run in the background, it is set to a number greater than 0.	M/PACT Specifically: n Pre-Report triggers n Post-Report triggers

Entry Points to the %msql Routine

The %msql routine supports several utility entry points, which you may use in M programs to perform the following functions:

- Call Open M/SQL objects
- Emulate various aspects of form behavior
- Establish an Authorization ID

The following tables list and briefly describe these utility entry points to %msql.

Use the following entry points to call Open M/SQL objects from an M program:

Table 11-2: Entry Points For Calling Open M/SQL Objects

Entry Point	Function
orm^%msql	Calls a form
menu^%msql	Calls an old-style menu
query^%msql	Calls a pre-defined query
report^%msql	Calls a report

Use the following entry points to emulate various aspects of form behavior:

Table 11-3: Entry Points For Emulating Form Behavior

Entry Point	Function
help^%msql	Displays scrolling help text
write^%msql	Writes a message on the screen
$$read^%msql	Reads a data value
windcln^%msql	Cleans up screen by erasing the residual display of windows after they have been exited

Use the following entry point to establish an Authorization ID (identify a user to Open M/SQL):

Table 11-4: Entry Point for Establishing an Authorization ID

Entry Point	Function
setaid^%msql	Establishes an Authorization ID (identifies a user to Open M/SQL)

For more information about these entry points, refer to the sections found later in this chapter entitled “Calling Open M/SQL Objects from M Programs” on page 11-27, “Emulating Form Behavior from M Programs” on page 11-34, and “Establishing Authorization ID from Programmer Mode” on page 11-39.

Open M/SQL Globals

There are two groups of globals used by Open M/SQL, **object definition globals** and **percent (%) globals**.

Object Definition Globals

The object definition globals are located in each development directory/UCI/namespace.

Caution Do not delete these globals. Doing so will delete your object definitions.

Most of the object definition globals begin with the letter “m”.

Note InterSystems reserves the syntax ^m* for its own use.

The following table lists the Open M/SQL object definition globals and briefly describes the information stored in each:

Table 11-5: Open M/SQL Object Definition Globals

Global	Contents
^ROUTINE	Macro source and intermediate code routines (for Open M/SQL on ISM systems only)
^mroutine	Macro source and temporary intermediate code routines (for Open M/SQL on ISM systems only)
^UTILITY	Scratch information for utilities (for Open M/SQL on ISM systems only)
^mcompd	Object compile driver configuration definitions
^mconv	Information about which conversion programs have been run on the major objects in this directory
^mdd	Base table definitions and view definitions
^mddc	Information for compiled base tables
^mexpnew	Export/import definitions (for the new-style Export/Import utility)
^mexport	Export/import definitions (for the old-style Export/Import utility)
^mform	Form definitions
^mformc	Information for compiled forms
^mhelp	Information for the Open M/SQL Help Menu
^mlock	Locking information for rows
^mmenu	Old-style menu definitions
^mmisc	Routine name prefixes

Table 11-5: Open M/SQL Object Definition Globals (Continued)

Global	Contents
^mobject	Menu object definitions, help document definitions, help topic definitions, and video attribute definitions
^mpriv	Privilege allocation definitions for Open M/SQL objects
^mql	Pre-defined query definitions
^mreport	M/PACT report definitions
^mreportc	Information for compiled M/PACT reports
^mtemp*	Miscellaneous run-time information
^mterm	Saved terminal specifications to be restored upon exiting Open M/SQL
^mutil	Miscellaneous utility definitions
^mxdd	Base table definitions queued for export using the Export/Import utility
^mxform	Form definitions queued for export using the Export/Import utility
^mxreport	Report definitions queued for export using the Export/Import utility
^mxql	Query definitions queued for export using the Export/Import utility
^mxmenu	Old-style menu definitions queued for export using the Export/Import utility
^mxmenob	Menu object definitions queued for export using the Export/Import utility
^mxdoc	Help document definitions queued for export using the Export/Import utility
^mxhtop	Help topic definitions queued for export using the Export/Import utility

Object Definition Globals Located in Common Directory

The following object definition globals are also located in the common directory/UCI/namespace (/usr/msql/common):

- ^mdd
- ^mddc
- ^mform
- ^mformc
- ^mhelp
- ^mmenu
- ^mmisc
- ^mobject
- ^mpriv
- ^mreport
- ^mreportc

If you are using the Open M/SQL Relational Server, you will also find the following globals in the common directory/UCI/namespace (/usr/msql/common):

- ^mroutine
- ^mtemp

Caution Do not delete these globals from the common directory under any circumstances. If you do so, you will have to reinstall Open M/SQL.

Open M/SQL Percent (%) Globals

The percent (%) globals are located in the Open M/SQL System Manager's directory.

Caution Do not delete these globals from the System Manager's directory. If you do so, you may have to reinstall Open M/SQL.

Also, do not change the default global protection status of these globals, and do not edit these globals by hand (with the exception of %rakeys).

The following table lists the Open M/SQL percent (%) globals and briefly describes the information stored in each:

Table 11-6: Open M/SQL Percent (%) Globals

Global	Contents
^%RDE	Full Screen Editor information
^%dafmlog	Results of FileMan->Open M/SQL links and updates
^%mmsg	Information for the "Message of the Day"
^%mobject	System help objects
^%msql	Miscellaneous Open M/SQL information
^%muser	Open M/SQL UserName definitions. Default protection status allows users to read/write/delete this global. To edit the contents of this global, use the User Security Definition window
^%omc	Relational Gateway information
^%oms	Open M/SQL Server configuration information.The System Manager can edit the contents of this global by editing Server configuration parameters
^%qarmisc	Miscellaneous M/PACT information
^%rakey	Open M/SQL keyboard definition information
^%rakeys	Open M/SQL keyboard definition information. Default protection status allows users to read/write/delete this global. This is the only % global that you might have occasion to edit by hand (see the *Open M/SQL Database Administrator's Guide* for more information).
^%rde	Full Screen Editor information
^%rekey	Open M/SQL keyboard definition information
^%sys	Open M/SQL system information
^%task	Open M/SQL Task Manager information. Default protection status allows users to read/write/delete this global. To edit the contents of this global, use the M/SQL Task Queue window

Inserting Code into Open M/SQL Applications

Open M/SQL provides a variety of locations where you may directly insert M code or SQL code into an application. This enables you to customize the application to your exact specifications.

The following table lists the locations within the Open M/SQL application development environment where inserted code is supported and shows what type of code you can insert:

Table 11-7: Locations Where M/SQL Permits Inserted Code

Location	Type of Code
Triggers in base tables (Action Types: M Code, SQL Code, Routine)	M code lines SQL statement
Triggers in forms (Action Types: M Code, SQL Code, Routine, Set Field)	M code lines SQL statement
Triggers reports (Action Types: M Code, Routine)	M code lines
Override queries for base table and form lookups	SQL WHERE and ORDER BY statements
Computed fields in base tables, forms, and reports	M code lines M IF conditions
Internal-to-External and External-to-Internal Conversion Code in base tables and forms	M code lines
Additional Validation Code in base tables and forms	M code lines
Conditions for Required-Maybe fields in base tables and forms	M IF conditions
Conditions for executing variable window placement	M IF conditions
Expressions for map subscripts and pieces	M expressions
NEXT Subroutine in maps	M code lines
Conditions for conditional maps	M IF conditions
General Mapping	M code lines
Menu object options (Action Types: M Code, SQL Code, Routine)	M code lines SQL statement
Code to use mnemonic as accelerator in stand-alone menu bar attributes	M expressions
Conditions for exiting menu bar after invoking menu option	M expression
Conditions for setting the Active flag for a menu option	M expression
Conditions for suppressing display of a menu option	M expression

Table 11-7: Locations Where M/SQL Permits Inserted Code (Continued)

Location	Type of Code
Defining the Caller ID for a menu action	M expression
Conditions for executing variable menu placement (for menu objects only)	M IF conditions
Code to clear the screen before menu option invocation and restore the screen after option invocation	M expression
Conditions for exiting the Pop-Up Menu after invocation of a menu option	M expression
Conditions for collapsing all Pop-Up Menus, or the current Pop-Up Menu after the menu option is invoked	M expression
Menu options (for old-style menus) Action types: M Code, Routine	M code lines
Specification for Row ID for the form to call (when the target form is a single-row form)	M expression
Specification for the parent reference (when the source table of the target form is a child table with a known parent)	M expression

The types of inserted code referenced in the **Type of Code** column in the preceding table have the following meanings:

- **M code** — consists of one or more lines of valid M commands and arguments, without tags.
- **M expression** — M code that contains no commands and evaluates to a value.
- **M IF condition** — one or more M expressions separated by commas, as would follow an M IF command.
- **SQL statement** — SQL code that performs a single operation, e.g. SELECT, INSERT, UPDATE, DELETE, etc.

Inserted Code Can Reference Fields

Inserted M code can reference single-line as well as multi-line fields in the current base table.

To reference single-line fields, use the curly brace syntax, as follows:

```
{fieldname}
```

To reference multi-line fields, use the following syntax:

```
{fieldname(expression)}
```

This syntax requires that you specify an expression to indicate which line of the multi-line field you are targeting.

For example, the following code:

```
SET AddLine2={Address(2)}
```

sets the variable “AddLine2” to the second line of the “Address” field.

The expression cannot contain any of the following:

- Other fields
- The wildcard character “*”
- Curly braces

Open M/SQL does not check the expression for correctness. An incorrect expression may cause the code to fail.

Inserted Code Can Reference Variables and Globals

Inserted M code can reference local variables and globals. For each type of inserted code, certain percent (%) variables are uniquely applicable, while others are not applicable. This topic is discussed in more depth in the following sections.

To avoid potential conflicts with system code, your inserted M code should not reference variables with all-lowercase names, unless the inserted code is a trigger action (and therefore called as a subroutine) and protects its variables with a NEW command.

Anywhere you can insert M code you can also insert macro source code.

Open M/SQL Performs Syntax Checking on Inserted M Code

Open M/SQL automatically performs syntax checking on all inserted M code throughout the Open M/SQL environment, with the exception of the General Mapping facility in the Data Dictionary (where you may define a customized physical structure for a base table).

Whenever you exit a line of inserted M code, Open M/SQL checks it for proper syntax and reports any errors at the bottom of the screen.

Open M/SQL does not check SQL code for proper syntax.

Note Open M/SQL supports syntax checking for ISM systems only.

Triggers

Triggers are sequences of actions that you may define to automatically take place at various points during the execution of an Open M/SQL application.

You may define triggers at the following levels within the Open M/SQL application development environment:

- Base table
- Form
- Report

Trigger Action Types

A trigger may consist of one trigger item or a sequence of multiple trigger items.

Each trigger item must have an action type, which specifies the kind of action the trigger item will execute. The set of action types available to a trigger definition depends on where in the Open M/SQL environment the trigger is located.

For each trigger item, you may define an associated condition in the form of an M IF expression to control its execution. When the condition associated with a trigger item evaluates to TRUE, the specified trigger action executes. When the condition associated with a trigger item evaluates to FALSE, the specified trigger action does not execute.

Base Table Triggers

Base table triggers are defined in the Data Dictionary. They are sequences of database actions initiated by an INSERT, UPDATE, or DELETE action performed on a base table. These triggers help to maintain integrity constraints and other data dependencies.

There are six types of base table triggers, including:

- Pre-filing INSERT Triggers
- Pre-filing UPDATE Triggers
- Pre-filing DELETE Triggers
- Post-filing INSERT Triggers
- Post-filing UPDATE Triggers
- Post-filing DELETE Triggers

Base Table Trigger Action Types

Base table triggers support the following action types:

- M Code
- Routine
- SQL code

Each of these action types permits you to insert your own code.

Form Triggers

Form triggers extend the conventional notion of triggers to form, window, and field interaction. They are defined in the Form Generator to perform various operations tailored to the needs of the application end-user. Form triggers can execute at the form, window, and field levels of an application and are often used to define the control structure for application processing.

There are nine types of form triggers, including:

- Pre-Form Triggers (one per form)
- Post-Retrieval Triggers (one per form)
- Post-Form Triggers (one per form)
- Pre-Row Triggers (one per form, for multi-row forms only)
- Post-Row Triggers (one per form, for multi-row forms only)
- Pre-Window Triggers (one per window)
- Post-Window Triggers (one per window)
- Pre-Field Triggers (one per field)
- Post-Field Triggers (one per field)

When used to file data, a form invokes the filing operations of base tables, thus causing base table triggers to be pulled.

Form Trigger Action Types

Form triggers support a very long list of action types, many of which emulate very specific programming functions (e.g., invoke a specified form, window, or menu). Most form trigger action types do not permit you to insert your own code.

For a complete list of all action types supported by form triggers, see the *Open M/SQL User Interface Programming Guide*.

The form trigger action types that permit you to insert your own code include:

- M Code
- SQL Code
- Routine
- Set Field (which lets you set a specified field equal to an M expression).

Report Triggers

Report triggers are defined in M/PACT. They may be used to execute lines of M code, call pre-defined M routines, or invoke forms either before or after the execution of a report, with the resultant action usually depending on the evaluation of some M/PACT percent (%) variable.

There are two types of report triggers:

- Pre-Report Triggers
- Post-Report Triggers

Pre-Report triggers are pulled before the report runs, but after the report output device is opened and used. Any output from a Pre-Report trigger is sent to the output device for that run of the report.

Post-Report triggers are pulled after the report runs, but before the device has been closed, so that output from a Post-Report trigger is also sent to the output device for that run of the report.

Report Trigger Action Types

Report triggers support the following action types:

- M Code
- Routine
- Form

The M Code and Routine action types permit you to insert your own code.

You should only the Form action type if you intend to send the report to a terminal.

Referencing Fields in a Trigger

When defining triggers, you may reference any fields associated with the current base table in the following places:

- M IF condition expression associated with any trigger item
- M Code window for a trigger item of action type M Code
- Set Field window for a trigger item of action type Set Field
- Parameters to the routine specified for a trigger item of action type Routine

To reference a field, enclose the field name in curly braces, for example:

```
SET NAME={Name}
```

sets the M local variable “NAME” equal to the (internal) value of the “Name” field in the current base table.

Note The syntax {Table.Field} is not allowed in triggers.

Referencing Open M/SQL Percent (%) Variables in Triggers

When defining triggers, you may reference Open M/SQL percent (%) variables in the following places:

- M IF condition expression of any trigger item
- M Code window of a trigger item of action type M Code

The Open M/SQL percent (%) variables store information about system mode and recent user actions. You may use them to set up execution conditions for navigation actions or other programmer-defined operations.

For a complete list of the Open M/SQL percent (%) variables, see Table 12-1, “Open M/SQL Percent (%) Variables”.

Many percent variables appear enclosed in curly braces, a syntax used to denote internally stored system information. These variables are interpreted by Open M/SQL at run time.

A few of the percent variables you may find most useful when defining triggers are:

- {%action}
- {%filetype}
- {%presave}
- {%savedata}
- {%timeout}
- {%caller}
- {%inquiry_mode}
- {%retrieved}
- {%parent_reference}

For example, the M IF condition expression of a trigger item might test the value of the {%presave} variable, which is 1 if the current row has been filed, 0 otherwise.

Open M/SQL also supports the a list of return prefix percent variables that correspond to certain non-prefixed percent variables, as shown in the table below:

Table 11-8: Return Prefix Percent Variables

Percent Variable	Return Prefix Percent Variable
{%action}	{%return_action}
{%filetype}	{%return_filetype}
{%presave}	{%return_presave}
{%savedata}	{%return_savedata}
{%timeout}	{%return_timeout}

Whereas the non-prefixed percent variables store current values, the return prefix percent variables store the same values for the form that was last exited (i.e., the values that were active at the time of the last Post-Form Trigger).

Triggers can also make use of other Open M/SQL variables for a variety of purposes. For more information on defining triggers, see the *Open M/SQL Data Dictionary Guide* (base table triggers) or the *Open M/SQL User Interface Programming Guide* (form triggers).

Using SQL Code in a Trigger

A trigger item of action type SQL Code may contain a single SQL statement, occupying one or more lines. The SQL statement can be any of the following:

- SELECT
- INSERT
- UPDATE
- DELETE
- GRANT
- REVOKE
- %CHECKPRIV
- CREATE VIEW
- ALTER VIEW
- DROP VIEW

An example of a trigger item of action type SQL Code occurs in the pre-filing delete trigger for a parent table. This trigger item deletes all child rows belonging to a parent row when the parent row is deleted. It uses the following SQL syntax:

```
DELETE FROM ChildTable
WHERE ChildTable.ParentRef = :parid
```

where "parid" is a variable set to the Row ID of the parent table currently being deleted. This trigger item ensures referential integrity by making sure that there are no orphan child rows.

Note When entering SQL code for a trigger item, you must insert a space character between consecutive lines that should be separated by a space. The SQL Code definition window does not embed a space when it wraps from one line to the next.

Using M Code in a Trigger

A trigger item of action type M Code may contain one or more lines of M code. Each line may be up to 255 characters in length.

The M code lines you define are called as a subroutine. Therefore:

- You may use a QUIT statement to exit the trigger item and move on to evaluate the next item (if one exists) in the sequence.
- You may use a NEW statement to protect local variables referenced in the trigger so that they do not affect the caller.

The M Code defined in triggers may contain DO statements and may therefore call user-defined M routines.

Override Queries for Lookups

Open M/SQL lets you define lookup specifications at both the base table and form levels.

When you define lookup specifications, Open M/SQL automatically generates a default SQL query for each lookup query you define. You can optionally modify this default SQL query to customize lookup selection logic to your exact specifications.

Specifically, you can modify the WHERE and ORDER BY clauses of each generated lookup query. When you modify a generated lookup query you create an override lookup query. You must always use valid SQL syntax when modifying a generated lookup query.

You can modify a generated lookup query to do the following:

- Change an exact match lookup query to a partial match, or vice versa,
- Use a different transformation function from the standard for a given data type (ALPHAUP, UPPER, or your own function),
- Make a lookup query more restrictive by excluding rows that do not satisfy some set of criteria,
- Modify the order in which the rows are displayed in the lookup box.

Computed Fields

A computed field is a field whose value is calculated by one or more lines of M statements. The M statements may reference other fields within the current base table using the curly brace syntax, and may use M $ functions. Computed fields may also include embedded SQL statements delimited by the ##sql preprocessor function. Open M/SQL lets you define computed fields for base tables, forms, and reports.

A computed field must always set itself equal to the value of the calculation specified in the M statement(s). You may reference the computed field itself by enclosing an asterisk in curly braces, {*}. For example:

```
SET {*}={Qty}*{Price}
SET {*}=$SELECT({City}="Dallas":"LOCAL",1:"REMOTE")
```

You may also define IF conditions associated with the computed field that cause the field to produce a NULL result. An IF condition consists of one or more M expressions separated by commas and can contain field names in curly braces. When an IF condition evaluates to true, the computed field is NULL.

Internal/External Conversion Code

You may define external-to-internal and internal-to-external conversion code for fields. This code converts field values between external display format and internal storage format. It consists of M statements that modify the value of %val.

External-to-Internal Conversion Code

External-to-internal conversion code finds the external value of a field in %val and sets %val to an internal value. In addition, if it detects a problem with the external value, it can set %ok to zero and %msg to an error message, as described below in the section entitled "Additional Validation Code". For example, the following conversion code translates a date from external format to internal ($H) format:

```
SET %DS=%val do INT^%DATE SET %val=%DN
IF %val=-1 set %ok=0,%msg="Invalid date"
```

the entry point INT^%DATE receives input from the variable %DS and sends output to the variable %DN.

Internal-to-External Conversion Code

Internal-to-external conversion code finds the internal value of a field in %val and sets %val to the external value. The following code translates a date from internal format to external format:

```
set %val=$zd(%val)
```

The variable %ok has no effect in internal-to-external conversion code.

Other Conversion Code Variables

Conversion code can also reference the following variables:

- %old — internal value of a field as it existed when last retrieved from the base table.
- %oldext — external value of a field as it existed when last retrieved from the base table.
- %new — internal value of a field as it currently appears.
- %newext — external value of a field as it currently appears.

For example, to prevent the field "WasSeen" from being changed from Yes to No, the conversion code for "WasSeen" might include the following line:

```
IF %oldext["Y",%newext["N" set %ok=0,%msg="Can't..."
```

Additional Validation Code

Open M/SQL automatically performs some validation checking of base table and form-only fields based on the specified data type and information associated with the data type. For example, Open M/SQL tests a date for valid syntax. It also tests date fields to make sure that an entry is not earlier than the earliest allowable date, as mandatorily specified in Date Field Definition window.

The author of a base table or form can specify additional validation code to further refine the scope of allowable entries. Additional validation code consists of M statements that may optionally reference field names using the curly brace syntax. The internal value of a field resides in the variable %val. Additional validation code can be used to reject an invalid value for %val by setting %ok to zero and %msg to a text error message, as follows:

```
IF %val>$H SET %ok=0,%msg="No future dates please."
```

Additional validation code can reference the variables %old, %oldext, %new, and %newext, as described above in the section entitled "Internal/External Conversion".

Required-Maybe Fields

Required-Maybe is an option for base table and form-only fields that allows you to specify an M condition that determines whether or not the field is required. When the M condition you specify evaluates to TRUE, the field becomes required. When the M condition you specify evaluates to FALSE, the field becomes not required.

The M condition can be any sequence of M expressions separated by commas. It can also contain field names using curly brace syntax.

For example, suppose you want the base table field "Employees.LicenseNumber" to become required whenever the field "Employees.ParkingSpot" is not null. To do this, you can make "Employees.LicenseNumber" a Required-Maybe field based on the status of the "Employees.ParkingSpot" field, by specifying the following Required-Maybe condition:

```
{Employees.ParkingSpot}'=""
```

This condition evaluates to TRUE unless "Employees.ParkingSpot" is null or contains a null internal and external value separated by $C(1).

Map Subscripts and Pieces

In the Open M/SQL General Mapping facility where you map base tables to underlying global structures, you can specify subscripts and pieces in base table maps as M expressions. These M expressions may contain field names in curly braces.

For example, a constant subscript "DATA" is specified as:

```
"DATA"
```

A subscript consisting of the value of the "Dept" field in the base table is specified as:

```
{Dept}
```

A subscript can also be an expression, such as:

```
$select({Dept}="SALES":"MARKETING",1:{Dept})
```

Open M/SQL also allows master map subscripts to be expressions. In master map subscripts, fields in curly braces {} are limited to the RowID and fields on which the RowID is based.

See the *Open M/SQL Data Dictionary Guide* for more information on master map subscripts.

NEXT Subroutine

If you want a particular map subscript to be traversed by M code other than the default $ORDER on the subscript value, you can define an override NEXT subroutine for that subscript. A NEXT subroutine consists of one or more lines of M code that set the value of:

```
{Li}
```

where *i* is the number of the subscript.

You can skip a subscript value by executing a GOTO the label "NEXT". For example, the following NEXT subroutine for subscript 2:

```
SET {L2}=$ORDER(^({L2})) GOTO NEXT:{L2}#100=0
```

skips values that are multiples of 100.

Since a null subscript value indicates that there are no more rows, you should not use a null condition for the NEXT test. Also, do not use an IF command to perform the test.

The NEXT subroutine can execute a QUIT statement and use a NEW statement to protect any local variables that it uses.

Conditional Map

A conditional map is a map that Open M/SQL applies to base table data if a specified M condition evaluates to true.

The M condition consists of one or more M expressions separated by commas and can contain field names in curly braces.

For example, suppose you wanted to create an index of female patients only. You might define an IF condition for a conditional map, as follows:

```
IF {Patients.Sex}="F"
```

Calling Open M/SQL Objects from M Programs

The following sections describe how to call forms, reports, queries, menu objects, and old-style menus from within an M program or directly from the M programmer prompt using the M call syntax.

The M call syntax lets you invoke Open M/SQL objects from any of the following locations:

- From a trigger of action type M Code
- From menu option of action type M Code
- From within an M routine
- From the M programmer prompt

Calling Forms

Programmer-defined M code, including routines and inserted M code, can invoke Open M/SQL forms using the M form call syntax. There are two ways to use the form call syntax:

- Using the form's name or internal identification number
- Using the form's routine prefix

When calling forms that are compiled as NEW, you may use either syntax.

When calling forms that are compiled as OLD, you must use the name/ID# syntax.

Calling a Form By Its Name/ID#

To call a form by its form name or internal identification number, use the following syntax:

```
do form^%msql(form,user,rowid,parentid,inquiry,noloop,
   update,callerid,default_array,currrent_array,
   edit_array,scroll)
```

where *form* is the name or internal identification number of the form you want to invoke. It is the only required parameter

Calling a Form By Its Routine Prefix

To call a form by its routine prefix, use the following syntax:

```
do ^<routine_prefix>1(rowid,user,parentid,inquiry,
   noloop,update,callerid,default_array,currrent_array,
   edit_array,scroll)
```

where <*routine_prefix*> is the routine prefix associated with the form you want to invoke and "1" is the system supplied identifier appended to the prefix of the entry point routine.

Note You may find a form's routine prefix in the Routine Prefix field of the form-level Advanced Features window for that form.

Form Call Syntax Parameters

Both versions of the form call syntax use approximately the same parameter list. The only difference is the order of the *rowid* and *user* parameters, which is switched.

The parameters are position-sensitive, which means that you must assign some value to the first parameter before you can assign a value to the second parameter, and to the second parameter before you can assign a value to the third parameter, etc.

A null value will mark a place.

You may close the parentheses after the last positively defined parameter, and Open M/SQL will assume null values for all remaining parameters.

The entire parameters list is optional. You may type empty parentheses () to skip the entire list.

The following table lists and describes the parameters associated with the M form call syntax:

Table 11-9: Parameters Associated with M Form Call Syntax

Parameter	Meaning
rowid	The Row ID of a row in the form's base table. This causes the form to invoke the specified row. If null, the row selection window will appear (for single-row forms).
user	This is a non-meaningful parameter; Open M/SQL no longer uses this information. Enter the null value ("") to skip this parameter. **Note:** Open M/SQL stores UserName information in the %msql variable.

Table 11-9: Parameters Associated with M Form Call Syntax (Continued)

Parameter	Meaning
parentid	If the target form is associated with a child table and the form is designed to be called from a known parent, this parameter must reflect the Row ID of a row in the parent table. Otherwise, it is ignored.
inquiry	Set this flag to 1 to specify that the form is to be run in inquiry mode. Set to 0 for data entry mode. 0 is the default.
noloop	Set this flag to 1 to turn form looping OFF. Set to 0 to turn form looping ON. 0 is the default. This parameter applies to single-row database forms only.
updatestatus	You may set this flag as follows: "u" — forces an update of a single row. If set to "u", you must specify the RowID of a current row or a parent reference using the "rowid" or "parentid" parameters (parameters 2, 4). "i" — forces row insert. If set to "i", the "rowid" parameter is optional, and Open M/SQL will assign it if none is specified. "d" — displays a form. If set to "d", you must specify the RowID or parent reference of the row you want to display, unless the form is a form-only form.
callerid	Set this flag to the name of a caller to activate specifications for variable window placement or other uses of the {%caller} variable.
default_array	Here you may specify an array (actual name is your choice) to pass in field defaults. For database forms, this applies exclusively to database fields. For non-database forms, it applies to form-only fields.
current_array	Here you may specify an array (actual name is your choice) that will hold current field values on file after the form is exited. For database forms, this applies exclusively to database fields. For non-database forms, it applies to form-only fields but only when the non-database form files data.
edit_array	Here you may specify an array (actual name is your choice) that will hold the original values of fields that were edited. For database forms, this applies exclusively to database fields. For non-database forms, it applies to form-only fields but only when the non-database form transacts filing.
scroll	Set this flag to 1 to activate roll-and-scroll mode, 0 or NULL for normal windowing mode. You can only activate roll-and-scroll mode if it is enabled for the current form in the form-level Advanced Features window.

Calling Reports

Programmer-defined M code, including routines and inserted M code, can invoke M/PACT reports using the M report call syntax, as shown below:

```
do report^%msql(report_name,user,device,format,batchsw)
```

where “report_name” is the only required parameter.

The parameters are position-sensitive, which means that you must assign some value to the first parameter before you can assign a value to the second parameter, and to the second parameter before you can assign a value to the third parameter, etc.

A null value will mark a place.

You may close the parentheses after the last positively defined parameter, and Open M/SQL will assume null values for all remaining parameters.

The table below lists and describes the parameters associated with the M report call syntax:

Table 11-10: Parameters Associated with M Report Call Syntax

Parameter	Meaning
report_name	Specify the name or internal report identification number of the report you want to run.
user	This is a non-meaningful parameter; Open M/SQL no longer uses this information. Enter the null value (“”) to skip this parameter. **Note:** Open M/SQL stores UserName information in the %msql variable.
device	Here you may optionally specify the name of any valid output device to which your system is linked.
format	Here you may optionally specify the name of an output format for the specified device.
batchsw	Set this flag to 1 to specify that the report is to be run in the background. Set to 0 to specify that the report is to be run in the foreground. The default is 0.

Calling Queries

Programmer-defined M code, including routines and inserted M code, can invoke Open M/SQL queries using the M query call syntax, as shown below:

```
do query^%msql(query_name,user,device,format,batchsw)
```

where “query_name” is the only required parameter.

The parameters are position-sensitive, which means that you must assign some value to the first parameter before you can assign a value to the second parameter, and to the second parameter before you can assign a value to the third parameter, etc.

A null value will mark a place.

You may close the parentheses after the last positively defined parameter, and Open M/SQL will assume null values for all remaining parameters.

The table below lists and describes the parameters associated with the M query call syntax:

Table 11-11: Parameters Associated with M Query Call Syntax

Parameter	Meaning
query_name	Specify the name or internal report identification number of the query you want to run.
user	This is a non-meaningful parameter; Open M/SQL no longer uses this information. Enter the null value (“”) to skip this parameter. **Note:** Open M/SQL stores UserName information in the %msql variable.
device	Here you may optionally specify the name of any valid output device to which your system is linked.
format	Here you may optionally specify the name of an output format for the specified device.
batchsw	Set this flag to 1 to specify that the query is to be run in the background. Set to 0 to specify that the query is to be run in the foreground. The default is 0.

Calling Old-Style Menus

Programmer-defined M code, including routines and inserted M code, can invoke old-style menus using the M menu (old-style) call syntax, as shown below:

```
do menu^%msql(menu_name,user,loginsw,callerid)
```

where “menu_name” is the only required parameter.

The parameters are position-sensitive, which means that you must assign some value to the first parameter before you can assign a value to the second parameter, and to the second parameter before you can assign a value to the third parameter, etc.

A null value will mark a place.

You may close the parentheses after the last positively defined parameter, and Open M/SQL will assume null values for all remaining parameters.

The table below lists and describes the parameters associated with the M menu call syntax:

Table 11-12: Parameters Associated with M Menu Call Syntax

Parameter	Definition
menu_name	Specify the name or internal report identification number of the old-style menu you want to invoke.
user	This is a non-meaningful parameter; Open M/SQL no longer uses this information. Enter the null value (“”) to skip this parameter. **Note:** Open M/SQL stores UserName information in the %msql variable.
loginsw	This is an optional flag that controls the appearance of the User Identification window. Set this flag to 1 to invoke the User Identification window before running the menu. Set it to 0 to skip the User Identification window. The default is 0.
callerid	You may optionally set this parameter to a particular CallerID to activate any conditions associated with the menu that are based on a certain value being held by the {%caller} variable.

Calling Menu Objects

Programmer-defined M code, including routines and inserted M code, can invoke menu objects in either of two ways:

- Using a menu call entry point
- Using the menu object’s routine name

Calling Menu Objects Using a Menu Call Entry Point

The menu call entry point uses the following syntax:

```
do menuobj^raznm3(imn,fmtype,tmtype)
```

The following table lists and describes the parameters associated with the menu call entry point:

Table 11-13: Parameters Associated with the Menu Call Entry Point

Parameter	Meaning
imn	This parameter is required. Enter either the internal identification number or the name of the menu object you want to invoke. **Note:** If you identify the menu object by its name, you must use quotation marks to delimit the string.
fmtype	This is an optional parameter that you may use to declare the type of menu from which the menu you are calling is being called. You may enter the following: 0 or NULL — Pop-Up Menu 1 — Menu Bar This is useful when you have programmed a menu to respond in certain ways depending on the menu type of its calling menu, a value which is stored in the {%menutype} system variable.
tmtype	This is an optional parameter that you may use to declare the mode in which you want to invoke this menu. You may enter the following: 0 or NULL — Pop-Up Menu 1 — Menu Bar

Calling a Menu Object By Its Routine Name

Alternatively, you may call a menu object directly from M code using its routine name. To do so, you must know the routine prefix for the menu object (the routine name is formed by the concatenation of the routine prefix and the number 1.)

When you call a menu object by its routine name, you may optionally pass in a value for the Caller ID.

To call a menu object by its routine name, use the following syntax.

```
do ^<routine_prefix>1(callerid)
```

For example, if your pop-up menu prefix is “mp1”, and you want to pass in a Caller ID of “Admissions”, you would issue the following:

```
> do ^mp11(“Admissions”)
```

Emulating Form Behavior from M Programs

Open M/SQL provides several utility entry points to the Open M/SQL routine %msql that allow programmer-defined M code to emulate the behavior of Open M/SQL forms.

The table below lists and describes these entry points to %msql:

Table 11-14: Entry Points For Emulating Form Behavior

Entry Point	Function
help^%msql	Displays scrolling help text in a box.
write^%msql	Writes a message on the screen.
$$read^%msql	Reads a data value.
windcln^%msql	Cleans up screen by erasing the residual display of windows after they have been exited.

Like the entry points for invoking Open M/SQL objects, these utility entry points allow you to pass certain required and non-required parameters.

In all cases, the parameters are position-sensitive, which means that you must assign some value to the first parameter before you can assign a value to the second parameter, and to the second parameter before you can assign a value to the third parameter, etc.

A null value will mark a place.

You may close the parentheses after the last positively defined parameter, and Open M/SQL will assume null values for all remaining parameters.

Displaying Help Text in a Help Text Box

You may use the help^%msql utility entry point to display one or more lines of help text in a bordered box at a specified location on the screen.

This entry point uses the following syntax:

```
do help^%msql(helpref,x,y)
```

where “helpref” is the only required parameter.

The table below lists and describes the parameters associated with the help^%msql utility:

Table 11-15: Parameters Associated with help^%msql Utility

Parameter	Meaning
helpref	Here you must specify a reference to the variable or global node that contains the lines of help text you want to display.
x,y	Here you may optionally specify the coordinates for drawing the upper left corner of the help box. The value of x can range from 1 to 80, and the value of y can range from 1 to 24. The upper left corner of the screen is represented by the coordinates (1,1). The x and y coordinates must both be non-null, or they must both be null. If they are both null, the utility uses the current cursor position, as reflected in the variables $x and $y, i.e. x=$x+1, y=$y+1.

The help^%msql utility automatically draws a box that is large enough to accommodate all the help text that you provide, unless there is insufficient space on the screen below the starting position of the box to accommodate the help text. If there is insufficient space, the help text will scroll.

In a scrolling help text box, users may use the <UP ARROW> and <DOWN ARROW> as well as <PREVIOUS SCREEN> and <NEXT SCREEN> keystrokes to navigate the help text lines.

For example, the following call to help^%msql:

```
>do help^%msql("^rh(12,6)",20,10)
```

displays help text lines as follows:

```
^rh(12,6,1)=first line
^rh(12,6,2)=second line
...
```

in as large a box as necessary with its upper left corner at screen position (20,10).

Writing Message Text

The write^%msql utility entry point writes a message to the screen using the same conventions for timeout and erase as used by Open M/SQL forms.

This entry point uses the following syntax:

```
do write^%msql(message,timeoutsec,erasesw,x,y)
```

where “message” is the only required parameter.

The table below lists and describes the parameters associated with the write^%msql utility:

Table 11-16: Parameters Associated with write^%msql Utility

Parameter	Meaning
message	Here you must specify the text of the message you want to write to the screen. You must enclose the message text in quotation marks.
timeoutsec	Here you may optionally specify a number of seconds to wait for a read after the message is printed. If you set the timeout to a number greater than 0, the cursor will pause for the specified number of seconds (or until the user presses a key), and then return. If you set the timeout to 0 or null, the cursor will return immediately after printing the message.
erasesw	Here you may set a flag to specify whether the utility should erase the message before returning. Set this flag to 1 to erase the message before returning. **Note:** You should use this parameter in conjunction with a timeoutsec value greater than 0.
x,y	Here you may optionally specify the coordinates for the cursor position at which to start the message. The value of x can range from 1 to 80, and the value of y can range from 1 to 24. The upper left corner of the screen is represented by the coordinates (1,1). The x and y coordinates must both be non-null, or they must both be null. If they are both null, the utility uses the current cursor position, as reflected in the variables $x and $y, i.e. x=$x+1, y=$y+1.

Reading Fields

The $$read^%msql utility entry point mimics the behavior of a field on an Open M/SQL form. This function displays a text caption and a data entry slot of definable length to the screen at specified coordinates. It then reads in the data value input by the user. It supports both editing and horizontal scrolling capabilities.

This entry point uses the following syntax:

```
set <glvn>=$$read^%msql(prompt,x,y,scrlen,maxlen,default)
```

where <*glvn*> is a global or variable name and “prompt” is the only required parameter.

The table below lists and describes the parameters associated with the $$read^%msql utility:

Table 11-17: Parameters Associated with $$read^%msql Utility

Parameter	Meaning
prompt	Here you may specify the text of the caption that precedes the field's data entry slot. This is a required parameter, though you may set it to null to not display a caption. **Note:** Open M/SQL automatically draws the data entry slot using underscores.
x,y	Here you may optionally specify the coordinates for the cursor position at which to start displaying the caption text. The value of x can range from 1 to 80, and the value of y can range from 1 to 24. The upper left corner of the screen is represented by the coordinates (1,1). The data entry slot begins one space after the prompt or, if prompt is null, at the specified (x,y) coordinates. The x and y coordinates must both be non-null, or they must both be null. If they are both null, the utility uses the current cursor position, as reflected in the variables $x and $y, i.e. x=$x+1, y=$y+1.
scrlen	Here you may optionally specify the length of the field's data entry slot on the screen. This value cannot exceed the value for the maxlen parameter. It also cannot exceed the value of 80 minus the length of the prompt parameter. If this value is less than the value for the maxlen parameter, the field allows horizontal scrolling. If this value is null, maxlen is assumed to be null also.
maxlen	Here you may optionally specify the maximum length for a field value.
default	Here you may specify an optional default value for the field.

Cleaning Up Windows

You can use the Window Cleanup function (windcln^%msql) to erase windows from the screen after they have been exited. This function helps to resolve problems related to the residual display of window graphics after a window has been exited.

In Open M/SQL, whenever a user exits a window, the system schedules the window to be erased at the next opportune moment. Opportune moments normally occur before a new window is painted, or when the cursor is sitting on a prompt waiting for user input. While this is sufficient in most situations, sometimes it causes the erasure of windows to be delayed. This can be especially troubling if you are using M code or routines as an intermediary between Open M/SQL objects.

For example, suppose a menu object calls an M routine, and in turn the M routine makes calls to various form routines. As each of the forms is exited, it is possible for residual instructions that erase windows on the form to be left unresolved until such time as another form is called. This can cause strange behavior, such as windows transiently flashing on the screen, or portions of windows from the previously called form remaining unerased.

The Window Cleanup function erases all window graphics associated with any object that has been exited and is, therefore, waiting in the erase queue.

The Window Cleanup function uses the following syntax:

```
do windcln^%msql
```

There are two cases after a window/menu object is exited where the Window Cleanup function does not have any effect:

1. The window/menu object is set to not erase upon exit
2. A user exits via the GETOUTALL action

In these cases, Open M/SQL does not schedule any windows for erasure, therefore the Window Cleanup action has no effect.

Note Whenever you call a form or menu object directly from M code, you should always check the value of the %r variable upon returning to the M routine. If %r>103, this indicates that the user exited via a GETOUTALL action, in which case the M routine should quit immediately. This is important because when a user exits via a GETOUTALL action, Open M/SQL does not schedule any windows to be erased. Therefore, if the M routine calls another form or menu object, it may encounter unerased screen clutter from the previous form/menu object.

Establishing Authorization ID from Programmer Mode

Open M/SQL requires that each user have an Open M/SQL Authorization ID in order to use the system. The Authorization ID is a character string that identifies the current user to Open M/SQL and recalls the Open M/SQL privileges and setup parameters assigned to that user. All Open M/SQL security is based on the Authorization ID.

Open M/SQL stores the Authorization ID of the current user in the %msql variable.

Typically, users establish their Authorization IDs by entering a UserName and Password into the User Identification window, which Open M/SQL displays during login. However, users may also establish their Authorization IDs by setting %msql directly from M.

To pass the Authorization ID directly into %msql, issue a command using the following syntax at the M prompt before entering Open M/SQL:

```
> do setaid^%msql("<UserName>")
```

where <*UserName*> is a UserName that is registered in the User Table.

For example:

```
> do setaid^%msql("Zeus")
```

Note When setting the Authorization ID this way, you do not need to specify a password.

After issuing this command, user "Zeus" can log in to Open M/SQL by typing the following command:

```
> do ^%msql
```

If the value passed into %msql corresponds to a valid UserName, Open M/SQL acknowledges all development environment access privileges and object-level privileges assigned to that user.

If the value passed into %msql does not correspond to a valid UserName, Open M/SQL gives the user an Authorization ID of "unknown." Users classified as "unknown" are limited to minimal development environment access privileges (basically, run-time access only) and have object privileges only for those objects on which privileges are granted to user "_PUBLIC".

If a user passes his/her Authorization ID directly into %msql, Open M/SQL does not display the User Identification window at user login.

Intermixing Open M/SQL Objects with User-Defined M Routines

Open M/SQL provides a programmer utility called %msqlutl that helps to prevent potential conflicts between user-defined M routines and critical Open M/SQL variables.

Currently, %msqlutl supports three functions called by the following entry points:

- pushvars^%msqlutl
- popvars^%msqlutl
- $$msqlvars^%msqlutl

pushvars^%msqlutl

The pushvars^%msqlutl function protects critical Open M/SQL variables by pushing them out onto a stack.

You should use this entry point prior to invoking any user-defined M code that might potentially kill or overwrite some of these variables. You can call this entry point in either of two modes:

1. **Local Variables mode**

   In Local Variables mode, pushvars^%msqlutl pushes only those variables which are local to specific Open M/SQL objects, while leaving intact Open M/SQL global variables, which are needed to communicate between various Open M/SQL objects.

   To call pushvars^%msqlutl in Local Variables mode, use the following syntax:

   ```
   do pushvars^%msqlutl(0)
   ```

   Or, when running Open M/SQL on a DTM system, use the following syntax:

   ```
   do pushvars^%MSQLUTL(0)
   ```

   You may want to use this mode when your application is executing some combination of user-defined routines and Open M/SQL objects, as for example in the following sequence of events:

   ```
   User Routine --> Menu Object --> User Routine --> Form
   ```

2. **All Variables mode**

   In All Variables mode, pushvars^%msqlutl pushes not only the local Open M/SQL variables but the global variables as well.

   To call pushvars^%msqlutl in All Variables mode, use the following syntax:

   ```
   do pushvars^%msqlutl(1)
   ```

   Or, when running Open M/SQL on a DTM system, use the following syntax:

   ```
   do pushvars^%MSQLUTL(1)
   ```

   You may want to use this mode when your application is calling from an Open M/SQL object into a user-defined routine and does not invoke any other Open M/SQL objects, as for example in the following sequence of events:

   ```
   Menu Object -> UserRtn1 -> UserRtn2 -> UserRtn3
   ```

popvars^%msqlutl

The popvars^%msqlutl function reinstates the Open M/SQL variables which have been pushed out onto a stack.

You should use this entry point when returning from user-defined code after a previous call to pushvars^%msqlutl.

popvars^%msqlutl accepts no arguments.

To call popvars^%msqlutl, use the following syntax:

```
do popvars^%msqlutl
```

Or, when running Open M/SQL on a DTM system, use the following syntax:

```
do popvars^%MSQLUTL
```

$$msqlvars^%msqlutl

The $$msqlvars^%msqlutl function displays a list of critical Open M/SQL global variables which **must never be killed or overwritten** by user-defined M code.

You might use this function as an argument for an exclusive kill list, for example:

```
SET list=$$msqlvars^%msqlutl
KILL(list)
```

The list of critical Open M/SQL variables stored by the $$msqlvars^%msqlutl function includes the following:

```
db,%DIR,%dir,fro,h,kt,%is,lang,%mode,%msql,%mstack,%mw,
%mwo,%r,%rdsm,repaint,%trans,%volset,%wa,%wbvats,%we,
%wft,%wh,%wipe,%wmb,%wr,%ws
```

Contents of Applications

The remainder of this chapter describes the routines and globals that comprise base tables, forms, menu objects, old-style menus, reports, and queries.

Routine Names

Each form, menu object, report, or pre-defined query must be compiled before it can be run.

Note Old-style menus do not require compilation.

Furthermore, each base table associated with a form, report, or query must be compiled before the form, report, or query can be compiled.

Compilation generates a set of executable routines. These routines perform functions such as lookup and filing for a base table, drawing windows for a form, and printing output and calculating totals for a report.

Machine requirements limit the maximum size of a single routine. If more code is needed than can fit in a single routine, Open M/SQL generates multiple routines. To ensure compatibility across implementations, InterSystems recommends that you limit the maximum size of a single routine to 8KB, though this limit is not universal.

Each base table, form, menu object, report, and query has a routine prefix used to produce the names of all of its associated routines when it is compiled. The routine prefix is an alphanumeric name (leading character alphabetic) with a maximum length of 7 characters. When defining an object, you have the option of specifying a customized routine prefix or accepting the default routine prefix. Open M/SQL creates full routine names by appending a single character to this prefix. The following table shows the default routine name prefixes for all compilable objects:

Table 11-18: Default Routine Name Prefixes

Object	Default Prefix
Table	mt*n*
Form	mw*n*
Menu Bar	mb*n*
Pop-Up Menu	mp*n*
Report	mr*n*
Query	mq*n*

where *n* is a one-character unique identifier appended by the compiler.

The compiler generates the one-character unique identifier in the sequence 1-9, followed by uppercase A-Z, followed by lowercase a-z. This amounts to a maximum of 61 distinct routine names that can be associated with a particular base table, form, menu object, query, or report.

For example, given a base table with the routine prefix “rtname”, the compiler generates the following consecutive routine names:

```
rtname1
...
rtname9
rtnameA
...
rtnameZ
rtnamea
...
rtnamez
```

If you change the routine prefix of a base table, form, menu object, report, or query, you must be sure that the routine names generated from the new routine prefix do not cause conflicts with any existing user-defined routine names. Open M/SQL will not detect these conflicts.

For example, you should not assign the routine prefix “rdf” to any object in a directory that contains user-defined routine names such as “rdf1”, “rdff”, or “rdfX”, etc. You may, however, assign the routine prefix “rdf” to an object in a directory that contains a user-defined routine called “rdf11, since the compiler generates routine names by appending only one character to the routine prefix.

Note When you change a routine prefix, Open M/SQL does NOT delete the routines with names that begin with the old prefix.

Contents of Base Table Routines

The compiled routines for an Open M/SQL base table contain all of the following:

- Filing code for INSERT, UPDATE, and DELETE operations, including data type and other validation checking and internal/external conversion code.
- A compiled SQL query for retrieving the old values for a row, which is used during UPDATE operations to determine which values have changed. This query is called the Update Query for the base table.
- A compiled SQL query for each base table field specified as “Unique” in the base table definition, which is used to determine whether another row exists with an identical value. This query is called the Uniqueness Query for a base table field.

- Compiled SQL queries for default lookup specifications (row selection criteria), which are used by forms to select rows from the base table.
- Compiled M code or an SQL statement for each base table trigger item.

Entry Points to Primary Base Table Routine

The first compiled routine for a base table, whose name is the table's routine prefix concatenated with 1, contains three entry points to the filing operations for the base table (insert, update, and delete). These filing entry points operate on base table rows by calling the associated SQL statement, e.g. INSERT, UPDATE, or DELETE. You should not call these entry points directly from the M programmer prompt.

The tags for the three filing entry points are:

Table 11-19: Filing Entry Points to Primary Base Table Routine

Entry Point	Function
insert	M code that validates and then inserts a new row into the base table.
update	M code that validates and then updates a base table row.
delete	M code that validates and then deletes a row from the base table.

Base Table Definition Globals

Open M/SQL stores additional information about base tables in the ^mdd and ^mddc globals, as described in the following table:

Table 11-20: Storage of Base Table Information

Global	Contents
^mdd(1,irn)	Stores base table information, where irn is a unique identifying number for the base table.
^mddc(1,irn)	Stores information about the compiled base table routines.

Contents of Form Routines

The compiled routines for an Open M/SQL form contain all code necessary to run the form, including the following:

- Code to draw each window in the form and handle user interaction with data entry fields.
- Compiled SQL queries for lookup specifications (row selection criteria) to select rows from the base table associated with the form.
- Compiled M code or an SQL statement for each form, window, and field trigger.

Form Definition Globals

Open M/SQL stores additional information about forms in the ^mform and ^mformc globals, as described in the following table:

Table 11-21: Storage Of Form Information

Global	Contents
^mform(iform)	Stores form, window, and form field information for the form, where iform is a unique form identifier.
^mformc(iform)	Stores information about the compiled form routines.

Contents of Menu Object Routines

The compiled routines for an Open M/SQL menu object contain the following:

- Code to draw menu bars or pop-up menus and handle user interaction with data entry fields.

Menu Object Definition Globals

Open M/SQL stores additional information about menu objects in the ^mobject global, as described in the following table:

Table 11-22: Storage Of Menu Object Information

Global	Contents
^mobject(“object”, “menu”,imn)	Stores information about the compiled menu object routines.

Contents of Old-Style Menu Routines

Old-style menus do not require compilation; therefore Open M/SQL does not generate any compiled routines for them.

Open M/SQL stores all information about old-style menus in the global ^mmenu(*imenu*), where *imenu* is a unique menu identifier.

Contents of Report Routines

The compiled routines for an M/PACT report contain compiled SQL queries and the necessary code to calculate subtotals, totals, and all user-defined code (e.g. computed fields) and to print headers, trailers, and all data.

Report Definition Globals

Open M/SQL stores additional information about reports in the ^mreport and ^mreportc globals, as described in the following table:

Table 11-23: Storage Of Report Information

Global	Contents
^mreport(ireport)	Stores report information where ireport is a unique report identifier.
^mreportc(0,ireport)	Stores information about the compiled report routines.

Contents of Query Routines

The compiled routines for a pre-defined query created via the Query Generator or Interactive Query Editor contain compiled SQL queries and the necessary code to print the query results.

Open M/SQL stores additional information about the query in the global ^mql(*iquery*), where *iquery* is a unique query identifier.

CHAPTER

12

Open M/SQL Developer Utilities

Open M/SQL provides a group of utilities to help application programmers manage their Open M/SQL applications. These utilities are available on the Developer Utilities menu, which is accessible via an option on the Open M/SQL Main Menu.

This chapter describes how to use each of the utilities on this menu.

Specifically, it covers the following topics:

- Accessing the Developer Utilities Menu
- Using the Object Compile Driver Utility
- Checking the Integrity of Open M/SQL Objects
- Searching for Strings in Open M/SQL Objects
- Invoking Macro Source Routine Utilities
- Querying Objects by Routine Prefix
- National Language Reports

Accessing the Developer Utilities Menu

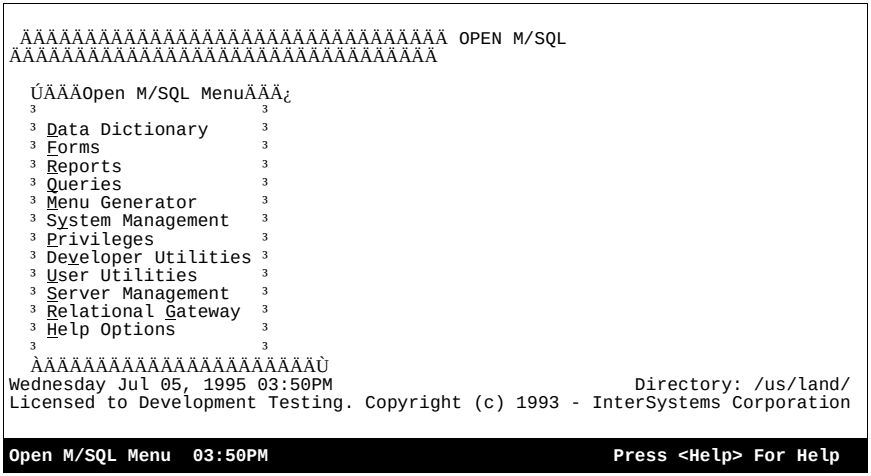
The Developer Utilities menu contains utilities that can help you manage your Open M/SQL applications.

Procedure **To access the Developer Utilities menu:**

1. **At the M programmer prompt, type the following command to enter Open M/SQL:**

   ```
   > do ^%msql
   ```

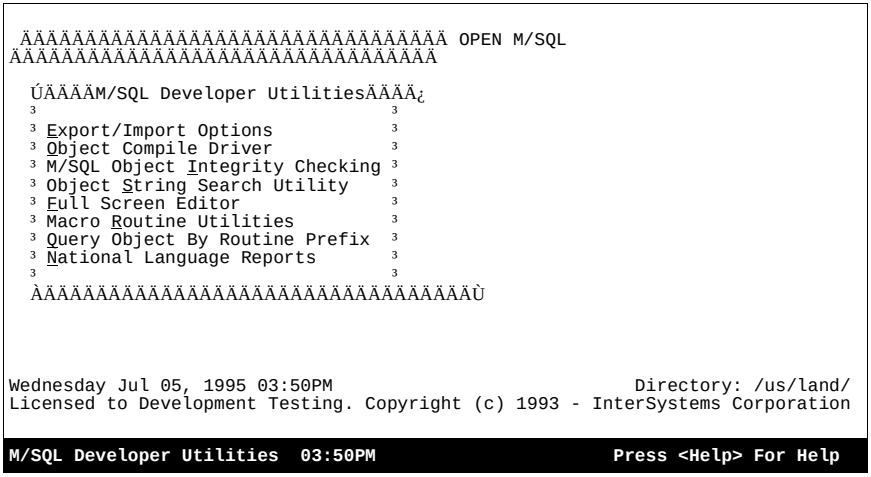
   You see the Open M/SQL Main Menu, as shown below:



2. **From the Open M/SQL Main Menu, select the Developer Utilities option.**

   Note: You may type *v* to select this option—it is a mnemonic accelerator.

You see the Developer Utilities menu, as shown below:



Note: The Database Administrator can restrict users from accessing the Developer Utilities menu by disabling the Developer Utilities option in the User Security Definition form.

The following table lists and describes the options located on the Developer Utilities menu:

Table 12-1: Options on Developer Utilities menu

Option	Description
Export/Import Options	Select this option to use the Export/Import utility (EIU), an automated, window-based utility that allows you to port Open M/SQL object definitions between different directories and different computers. We do not document the use of the Export/Import utility in this guide. For complete documentation on how to use this utility, see the *Open M/SQL Database Administrator's Guide*.
Object Compile Driver utility	Select this option to use the Object Compile Driver utility, which allows you to define compilation configurations for groups of multiple objects or entire applications. When you run a compilation configuration, it batch-compiles the code for all included objects.
Object Integrity Checker utility	Select this option to use the Object Integrity Checker utility. This utility checks the integrity of specified object definitions or groups of object definitions in the current directory and generates a report of all integrity errors that it finds. You can also set the Integrity Checker to automatically correct some of the integrity errors it finds.

Table 12-1: Options on Developer Utilities menu (Continued)

Option	Description
Object String Search utility	Select this option to use the Object String Search utility. This utility searches through specified object definitions for a specified text string.
Full Screen Editor	Select this option to enter the Full Screen Editor environment, where you may create and edit routines. When you exit the Full Screen Editor, you return to the Developer Utilities menu.
Macro Routine Utilities	Selecting this option invokes a submenu called Routine Utilities. The Routine Utilities submenu displays a list of Open M/SQL routine utilities. You may select any option on the Routine Utilities submenu to invoke and use the corresponding routine utility. When you exit the routine utility, you return to the Routine Utilities menu.
Query Object By Routine Prefix	Select this option to access the Object Routine Prefix utility. This utility lets you look up any Open M/SQL routine prefix to obtain information about where the routine is being used.
National Language Reports	Select this option to print the National Language Report, which displays the translations provided for all system-generated messages, key labels, menu titles, and menu option text throughout the Open M/SQL environment.

Using the Object Compile Driver Utility

The Object Compile Driver utility lets you define and store lists of Open M/SQL objects for serial compilation.

Using this utility, you can recompile large sections of your application or all of your application, with a single command.

To use the Object Compile Driver utility, you create and store compilation configurations. Each compilation configuration can include multiple objects of all of the following types:

- Base Tables
- Forms
- Menu Objects
- Reports
- Queries
- Routines

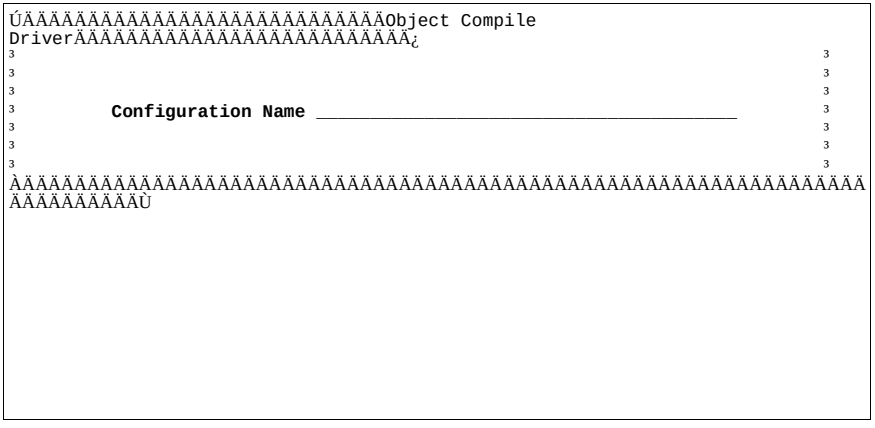
When you compile an Object Compile Driver configuration, you batch-compile all included objects.

The Object Compile Driver utility is available as an option on the Developer Utilities menu.

Procedure **To use the Object Compile Driver utility:**

1. **From the Developer Utilities menu, select the Object Compile Driver option.**

   You see the Object Compile Driver lookup window, as shown below:

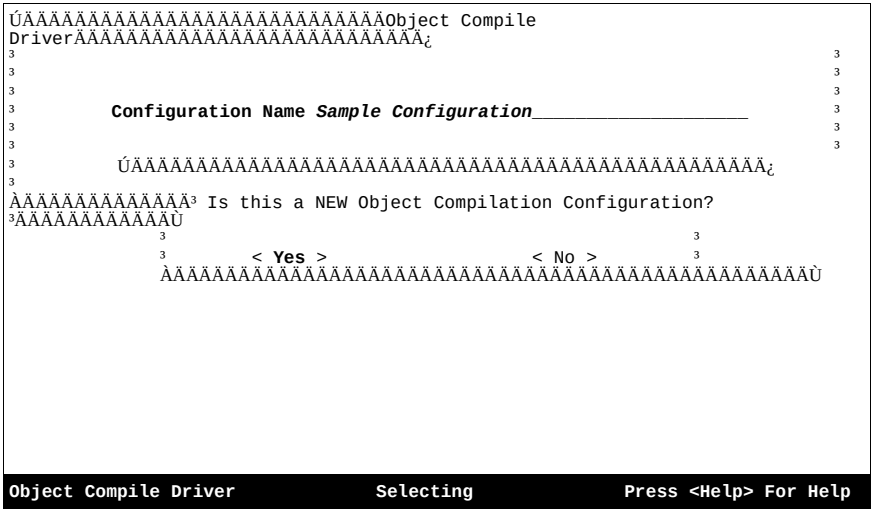


2. At the Configuration Name field in the Object Compile Driver lookup window, enter a configuration name, and press <RETURN>.

You can retrieve an existing compilation configuration to edit or create a new one.

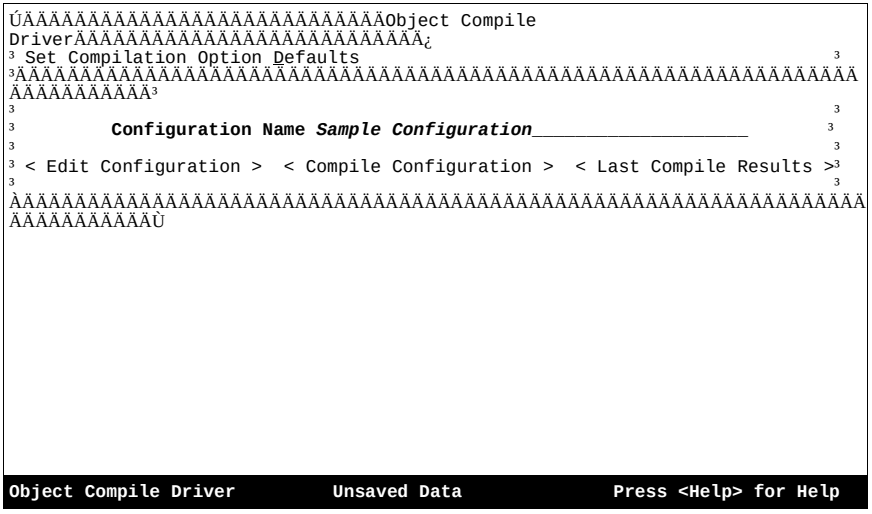
To retrieve an existing compilation configuration, enter a complete or partial name, and press <RETURN>. You see a lookup box that lists all matching entries. To see a lookup box that lists all existing entries, leave the Configuration Name field blank and press the <SEARCH CURRENT TABLE> key. Use the cursor positioning keys to navigate within the lookup box, and press <RETURN> to select an entry.

To create a new compilation configuration, enter a new configuration name (it must not match the name of any existing compilation configuration). You see the "Is this a NEW Object Compilation Configuration?" dialog box, as shown below: .



Here, you may create a new compilation configuration by pressing <RETURN> on the <Yes> action field.

When you have entered a configuration name (either new or existing), the Object Compile Driver window fills out with its complete set of fields, as shown below:



The table below lists and describes the fields located on the Object Compile Driver window:

Table 12-2: Fields on Object Compile Driver window

Field	Description
Configuration Name	Here you may edit the name specified for compilation configuration in the lookup window. Names may range from 1 to 40 characters in length and may contain any alphanumeric or punctuation characters, including underscores and blank spaces. This is a required field.
<Edit Configuration>	Press <RETURN> on this action field to access the Object Compile Driver Items window, where you may define the contents of your compilation configuration.
<Compile Configuration>	Press <RETURN> on this action field to run the compilation configuration.
<Last Compile Results>	Press <RETURN> on this action field to access the Object Compile Driver Results window, where you review the results from the last time the current compilation configuration was compiled.

Setting Compilation Option Defaults

For each compilation configuration you create, you may set up a series of compilation option defaults to control how Open M/SQL compiles the various objects in the list.

These compilation option defaults operate on a configuration-wide basis, i.e., they affect either all objects defined in the entire compilation list or all objects of a certain object type defined in the list.

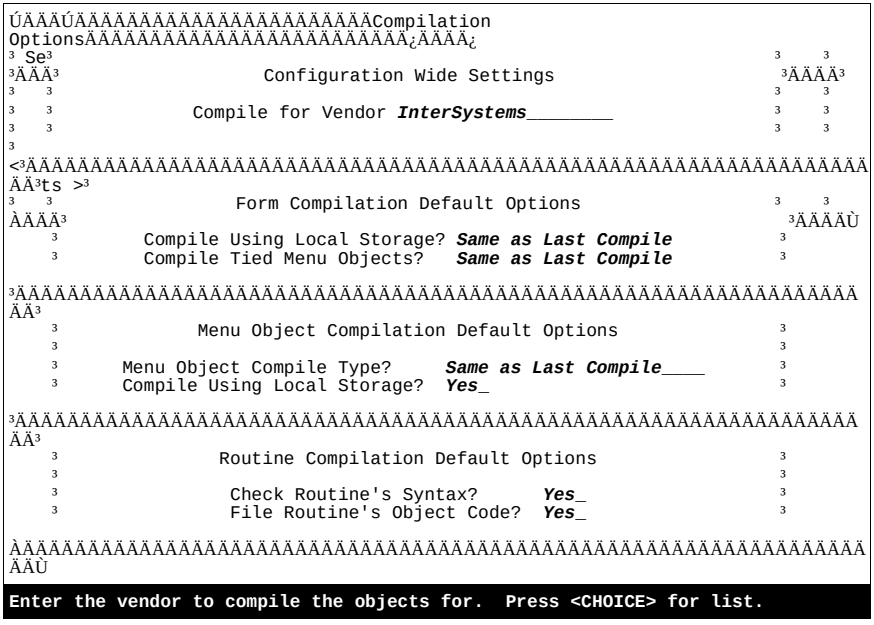
You can edit these compilation option defaults at any time.

Procedure **To set defaults for compilation options:**

1. **From the Object Compile Driver window, press the <ENHANCE><ENHANCE> keystroke sequence to access the Object Compile Driver menu bar.**
2. **From the Object Compile Driver menu bar, select the Set Compilation Option Defaults option.**

   Note: This is the only option on the Object Compile Driver menu bar.

   You see the Compilation Options window, as shown below:



The following table lists and describes the fields on the Compilation Options window:

Table 12-3: Fields on Compilation Options Window

Field	Definition	Objects Affected
Compile for Vendor	Here you specify the name of the M implementation for which you want to compile the objects in your compilation list. You may choose among the following M implementations: ▪ DSM ▪ DTM ▪ InterSystems ▪ MSM The host M system on which you are currently running is the default response. When you run a compilation configuration, Open M/SQL generates intermediate code for the M implementation you specify here. This option gives you the ability to compile the objects in your compilation list for an M implementation other than the one you are currently using. You can use this option to generate intermediate code for an application before moving it to a run-time environment on another M system. For example, you might want to develop your application on an InterSystems UNIX platform and run your application on a PC with DTM. To do this, you could build a compilation configuration for the complete application and compile it for DTM, then move the globals and the .INT code for the compiled routines from the UNIX machine to the PC.	All objects in list
Compile Using Local Storage?	The Local Storage option is designed to help you avoid situations where <STORE> errors occur during the compilation of a form due to the building of large queries. If you answer *No* to this field, Open M/SQL compiles all forms included in the compilation list using globals and not in local storage. This protects against <STORE> errors but slows compilation time. If you answer *Yes* to this field, Open M/SQL compiles all forms included in the compilation list using local arrays. This makes the compilation faster. When the forms you are compiling are not unusually large or complex, you can safely capitalize on the speed advantages of using local storage. If, however, you encounter a <STORE> error while compiling in local storage, you should change your response to *No*. If you answer *Same as Last Compile*, Open M/SQL compiles every form the same way it was last compiled. Same as *Last Compile* is the default response.	Forms only

Table 12-3: Fields on Compilation Options Window (Continued)

Field	Definition	Objects Affected
Compile Tied Menu Objects?	Here you may answer *Yes*, *No*, or *Same as Last Compile* to indicate whether or not you want Open M/SQL to recompile menu objects that are tied to the forms in your compilation list. In order to be recompiled under this option, a menu object must be explicitly tied to the form, not just attached to it. **Note:** This option only applies to forms that are compiled as NEW. *Same as Last Compile* is the default response.	Forms only
Menu Object Compile Type	Here you may specify how you want Open M/SQL to compile the menu objects in your compilation list. Press the <LIST CHOICES> key to see a list of the menu object compilation options from which you may choose. They include: ▪ **Pop-Up Menu** — Compile all menu objects as pop-up menus. ▪ **Menu Bar** — Compile all menu objects as menu bars. ▪ **Both Pop-Ups and Menu Bars** — Compile all menu objects as both pop-up menus and menu bars. ▪ **Same as Last Compile** — Compile all menu objects the same way as they were last compiled. The default option is *Same as Last Compile*.	Menu Objects only
Compile Using Local Storage?	Here you may answer *Yes* or *No* to indicate whether or not you want to compile the menu objects included in your compilation list using local storage. Answer *Yes* to compile menu objects using local arrays. This makes the compilation faster but may cause <STORE> errors to occur. Answer *No* to compile menu objects using globals. This prevents space allocation errors but makes the compilation slower. *Yes* is the default response.	Menu Objects only
Check Routine's Syntax?	Answer *Yes* or *No* to specify whether or not you want Open M/SQL to perform syntax checking on the routines included in your compilation list. *Yes* is the default response.	Routines only
File Routine's Object Code?	Answer *Yes* or *No* to specify whether or not you want Open M/SQL to file the object code generated by compiling the routines included in your compilation list. *Yes* is the default response.	Routines only

3. **In the Compile for Vendor field, specify the name of the M implementation for which you want to compile the objects in your compilation list.**

   The host M system on which you are currently running is the default response.

   Note: You may compile for an M implementation other than the one you are currently using.

4. **At the Use Local Storage During Compile field, answer *Yes*, *No,* or *Same as Last Compile* to indicate whether you want to compile the forms included in your compilation list using local arrays or globals.**

   This field applies only to forms being compiled as NEW.

   Same as Last Compile is the default response.

5. **At the Recompile Any Tied Menu Objects? field, answer *Yes*, *No,* or *Same as Last Compile* to recompile menu objects that are explicitly tied to the forms in your compilation list.**

   Same as Last Compile is the default response.

6. **At the Menu Object Compile Type field, select an option to indicate how you want Open M/SQL to compile the menu objects included in your compilation configuration.**

   You may press the <LIST CHOICES> key to see a lookup box that lists the menu object compilation options from which you may choose. They include:

   - *Pop-Up Menu*
   - *Menu Bar*
   - *Both Pop-Ups and Menu Bars*
   - *Same as Last Compile*

   Same as Last Compile is the default response.

7. **At the Compile Using Local Storage? field, answer *Yes* or *No* to indicate whether you want to compile the menu objects included in your compilation list using local arrays or globals.**

   Yes is the default response.

8. **At the Check Routine Syntax? field, Answer *Yes* or *No* to specify whether or not you want Open M/SQL to perform syntax checking on the routines included in your compilation list.**

   Yes is the default response.

9. **At the File Routine's Object Code? field, Answer *Yes* or *No* to specify whether or not you want Open M/SQL to file the object code generated by compiling the routines included in your compilation list.**

   Yes is the default response.

10. **When you finish defining the compilation options for your compilation configuration, press the <PROCEED> key to save your definitions and exit the Compilation Options window.**

Defining the Contents of a Compilation Configuration

Your compilation configuration can include multiple objects the following types:

- Base Tables
- Forms
- Menu Objects
- Reports
- Queries
- Routines

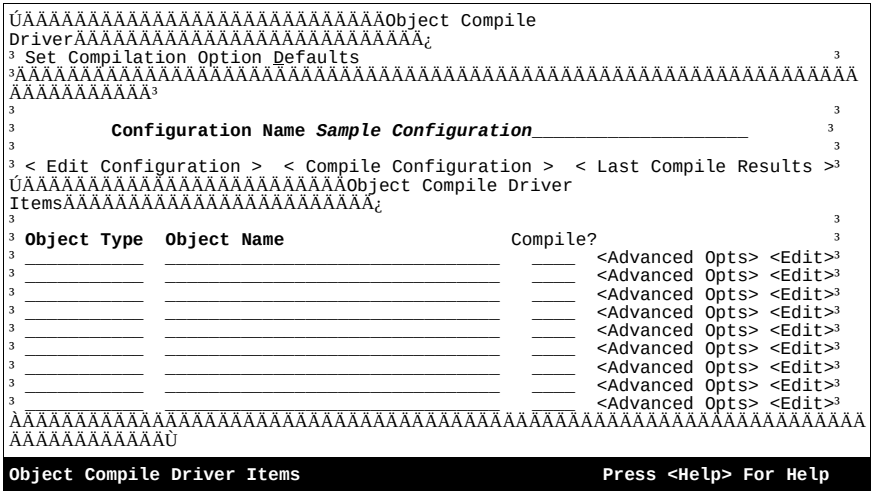
You may add as many objects to your compilation configuration as you wish.

You do not need to have privileges on an object to include it in your compilation configuration.

Procedure **To define the contents of a compilation configuration:**

1. **From the Object Compile Driver window, press <RETURN> on the <Edit Configuration> action field.**

You see the Object Compile Driver Items window, as shown below:



The table below lists and describes the fields located on the Object Compile Driver Items window:

Table 12-4: Fields on Object Compile Driver Items window

Field	Description
Object Type	Here you specify the type of the object you want to add to the compilation list. Press the <LIST CHOICES> key to see a list of the object types from which you may choose. They include: ▪ Base Tables ▪ Forms ▪ Menu Objects ▪ Reports ▪ Queries ▪ Routines
Object Name	Here you specify the name of the particular object you want to add to your compilation list. Press the <LIST CHOICES> key to see a list of all objects (of the specified object type) defined in the current database. **Note:** When specifying the name of a routine, you must explicitly specify a routine extension, either *.MAC* or *.INT*.

Table 12-4: Fields on Object Compile Driver Items window (Continued)

Field	Description
Compile?	Here you may enter *Yes* or *No* to indicate whether or not you want to compile the current object when you run the compilation configuration. Enter *Yes* to compile it, *No* to not compile it. *Yes* is the default response. You can change the setting of this switch back and forth to selectively include or exclude objects from the compilation list.
<Advanced Opts>	This field is only accessible if the object you are specifying is a form, menu object, or routine. For all other objects, the cursor cannot land on this field. When specifying a form, menu object, or routine, you may press <RETURN> on this action field to access a small popup window that lets you define several compilation options associated with the current object. These options override any default options specified in the configuration-wide Compilation Options window.
<Edit>	Press <RETURN> on this action field to access the object definition form for the current object. Within the object definition form, you may edit the definition of the current object however you wish. When you exit the object definition form, you return to the Object Compile Driver Items window.

Note: To skip ahead to the next field on the Object Compile Driver Items window, use the <TAB> key or the <RIGHT ARROW> key. To move back to the previous field, use the <LEFT ARROW> key.

2. **At the Object Type field, specify the type of the object you want to add to the compilation list.**

   You may press the <LIST CHOICES> key to see a lookup box that lists all object types from which you may choose.

3. **At the Object Name field, specify the name of the particular object you want to add to the compilation list.**

   You may press the <LIST CHOICES> key to see a list of all objects (of the specified object type) defined in the current database.

   Note: When specifying the name of a routine, you must explicitly specify a routine extension, either *.MAC* or *.INT*.

4. **At the Compile? field, enter *Yes* or *No* to indicate whether or not you want the current object to be compiled when you run the compilation configuration.**

   Yes is the default response.

5. **If the object you are currently entering is a form or menu object, you may press <RETURN> on the <Advanced Opts> action field to access a small popup window that lets you specify compilation options specifically associated with the specific object (see "Advanced Options for Compilation List Items" below).**

6. **You may press <RETURN> on the <Edit> action field to activate a direct link into the current object's definition form, where you may view or edit the definition (see "Editing an Object Definition" below).**

7. **Continue entering additional objects to the compilation list in the same way as described above.**

8. **When you have entered all objects that you want to be included in your compilation configuration, press the <PROCEED> key to save your definitions and exit the Object Compile Driver Items window.**

Advanced Options for Compilation List Items

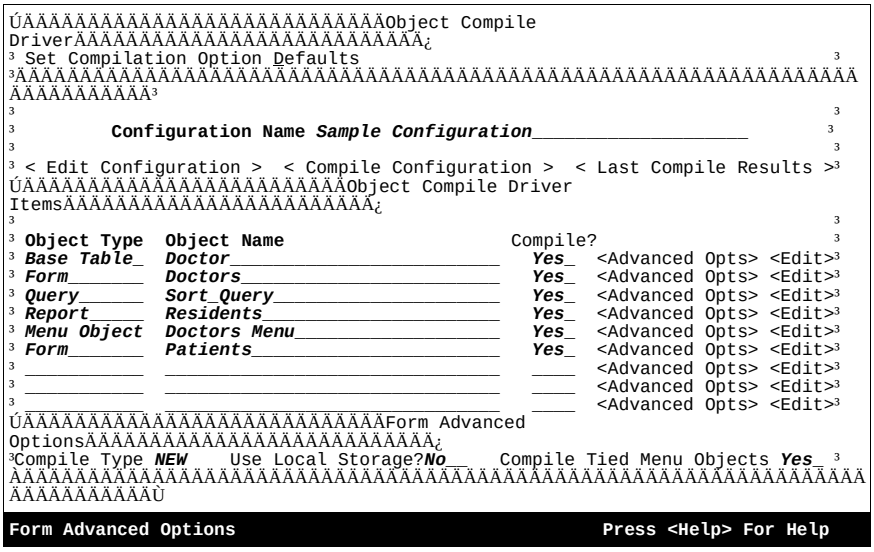
The <Advanced Options> action field on the Object Compile Driver Items window is only accessible if the object you are currently defining is a form, a menu object, or a routine. For all other objects, this field is skipped.

For forms, menu objects, and routines, you may use this option to access a window that lets you specify compilation options specifically associated with the current object.

Note The options you specify in this window override the configuration-wide compilation options specified in the Compilation Options window (accessed via the Object Compile Driver menu bar).

Advanced Options for Forms

When adding a form to the compilation list, you may press <RETURN> on the <Advanced Options> action field to access the Form Advanced Options popup window, as shown below:



The table below lists and describes the options located on the Form Advanced Options window:

Table 12-5: Options on Form Advanced Options window

Field	Description
Compile Type	This is an output-only field that shows the type of the last compilation, i.e., the version of the Forms Compiler last used to compile the form. This field can have either of the following two values: n NEW (post-Vesrion D version of the Forms Compiler) n OLD (pre-Version E version of the Forms Compiler) The cursor cannot land on this field.
Use Local Storage?	Here you may answer *Yes* or *No* to indicate whether or not you want to compile the current form using local storage. Answer *Yes* to compile the form using local arrays. This makes the compilation faster but may cause <STORE> errors to occur. Answer *No* to compile the form using globals. This prevents space allocation errors but makes the compilation slower. *No* is the default response.

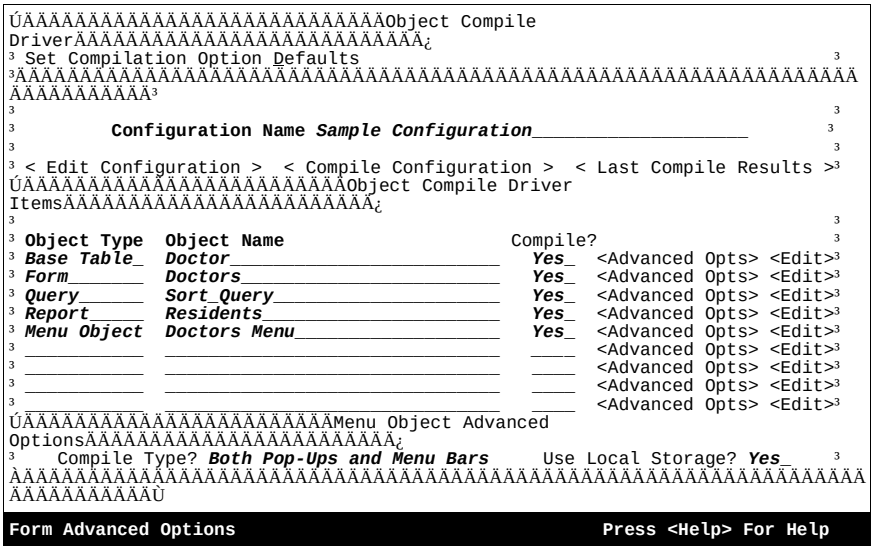
Table 12-5: Options on Form Advanced Options window (Continued)

Field	Description
Compile Tied Menu Objects?	Here you may answer *Yes* or *No* to indicate whether or not you want Open M/SQL to recompile any menu objects that are tied to the current form. In order to be recompiled under this option, a menu object must be explicitly tied to the form, not just attached to it. **Note:** This option only applies to forms that are compiled as NEW. *Yes* is the default response.

When you finish specifying compilation instructions on the Form Advanced Options window, press the <PROCEED> key to save your definitions and exit the window.

Advanced Options for Menu Objects

When adding a menu object to the compilation list, you may press <RETURN> on the <Advanced Options> action field to access the Menu Object Advanced Options popup window, as shown below:



The table below lists and describes the options located on the Menu Object Advanced Options window:

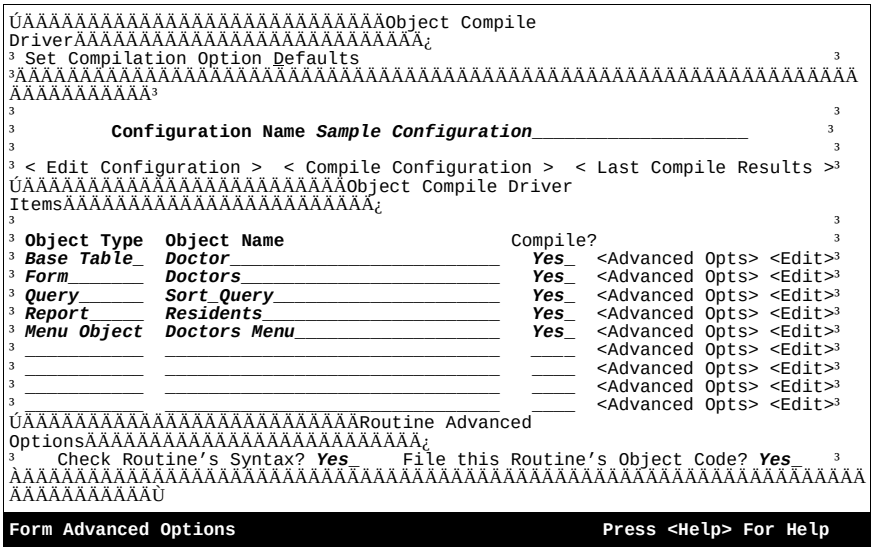
Table 12-6: Options on Menu Object Advanced Options window

Field	Description
Compile Type	Here you may specify how you want Open M/SQL to compile the current menu object. Press the <LIST CHOICES> key to see a list of the menu object compilation options from which you may choose. They include: n **Pop-Up Menu** — Compile all menu objects as pop-up menus. n **Menu Bar** — Compile all menu objects as menu bars. n **Both Pop-Ups and Menu Bars** — Compile all menu objects as both pop-up menus and menu bars. The default option is *Both Pop-Ups and Menu Bars*.
Use Local Storage?	Here you may answer *Yes* or *No* to indicate whether or not you want to compile the current menu object using local storage. Answer *Yes* to compile the menu object using local arrays. This makes the compilation faster but may cause <STORE> errors to occur. Answer *No* to compile the menu object using globals. This prevents space allocation errors but makes the compilation slower. *Yes* is the default response.

When you finish specifying compilation instructions on the Menu Object Advanced Options window, press the <PROCEED> key to save your definitions and exit the window.

Advanced Options for Routines

When adding a routine to the compilation list, you may press <RETURN> on the <Advanced Options> action field to access the Routine Advanced Options popup window, as shown below:



The table below lists and describes the options located on the Routine Advanced Options window:

Table 12-7: Options on Routine Advanced Options window

Field	Description
Check Routine's Syntax	Here you may answer *Yes* or *No* to indicate whether or not you want Open M/SQL to perform syntax checking on this routine when it is compiled. Answer *Yes* perform syntax checking. Answer *No* to skip syntax checking. *Yes* is the default response.
File this Routine's Object Code	Here you may answer *Yes* or *No* to indicate whether or not you want to file the object code associated withthis routine when it is compiled. Answer *Yes* to generate and file the object code for the routine. Answer *No* to generate only intermediate (.INT) code for the routine. *Yes* is the default response.

When you finish specifying compilation instructions on the Routine Advanced Options window, press the <PROCEED> key to save your definitions and exit the window.

Editing an Object Definition

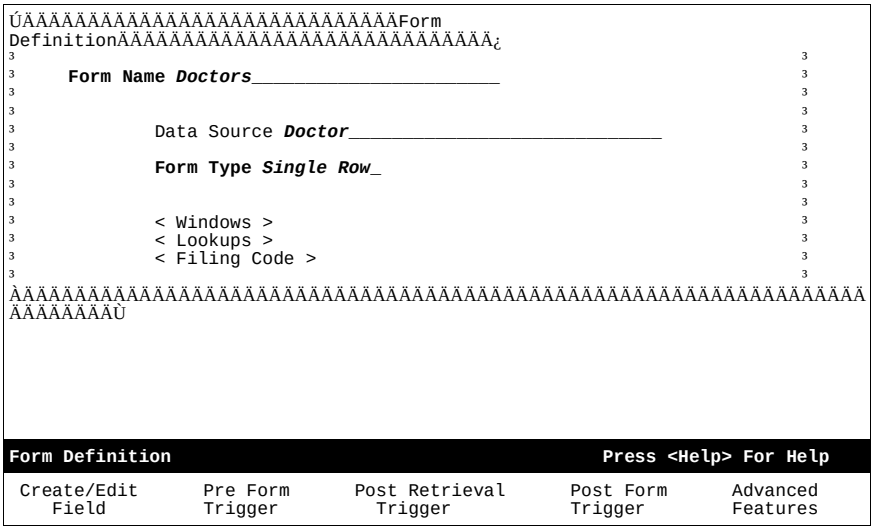
The Object Compile Driver utility lets you directly view or edit the definition of any object you include in the compilation configuration.

By pressing <RETURN> on the <Edit> action field you activate a direct link into the current object's definition form. Once you have accessed the object definition form, you may view or edit the definition.

Important You must have %ALTER privileges on an object in order to activate the direct link into its object definition form

When you exit a link-accessed object definition form, you return to the Object Compile Driver Items window, exactly where you activated the link.

For example, if you include the form "Doctors" in the compilation configuration and then you press <return> on the <Edit> action field, you see the Form Definition form for the "Doctors" form, as shown below::



Compiling the Configuration

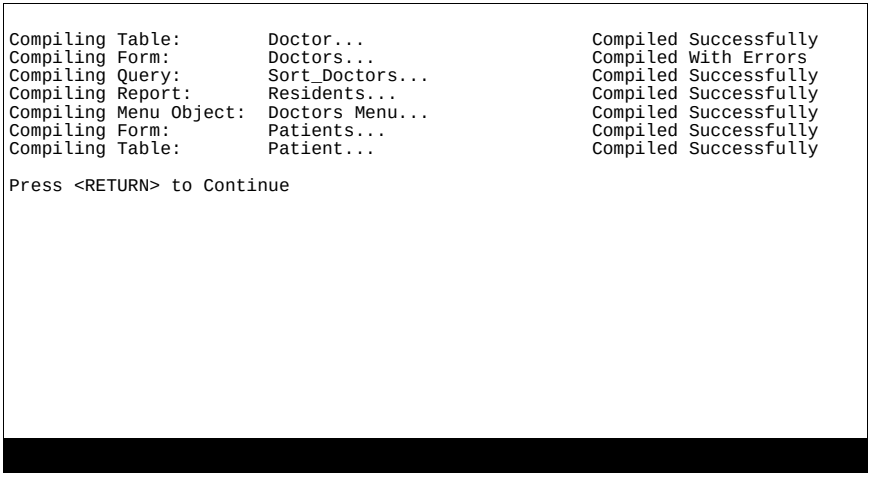
To compile a compilation configuration, press <RETURN> on the <Compile Configuration> action field.

Open M/SQL sequentially compiles each object in the list. While the compilation is happening, you see a compilation monitor window. For each object being compiled, the compilation monitor window displays the following information:

- Object Type
- Object Name
- Compilaton results message (either “Compiled Successfully” or “Compiled with Errors”)

When all objects in the list have been compiled, the compilation monitor window prompts you to press <RETURN>.

A sample compilation monitor window is shown below:



Press <RETURN> to return to the Object Compile Driver main window.

Viewing the Results of the Last Compilation

The <Last Compile Results> option on the Object Compile Driver main window lets you view the results from the last time the current compilation configuration was compiled.

Press <RETURN> on the <Last Compile Results> action field to access the Object Compilation Driver Results window, as shown below:

```
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄObject Compilation Driver
ResultsÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³                                                                              ³
³                                                      Compilation             ³
³ Object Type   Object Name                            Status                  ³
³ Base Table    Doctor                                 Success___  <View Err>³
³ Form          Doctors                                Failure___  <View Err>³
³ Query         Sort_Doctors                           Success___  <View Err>³
³ Report        Residents                              Success___  <View Err>³
³ Menu Object   Doctors Menu                           Success___  <View Err>³
³ Form          Patients                               Success___  <View Err>³
³ Base Table    Patient                                Success___  <View Err>³
³                                                      __________  <View Err>³
³                                                      __________  <View Err>³
³                                                      __________  <View Err>³
³                                                      __________  <View Err>³
³                                                      __________  <View Err>³
³                                                      __________  <View Err>³
³                                                      __________  <View Err>³
³                                                      __________  <View Err>³
³                                                      __________  <View Err>³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄCompilation Error
MessageÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
³ ___________________________________________________________________________ ³
ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
Object Compilation Driver Results       Inquiry         Press <Help> For Help
```

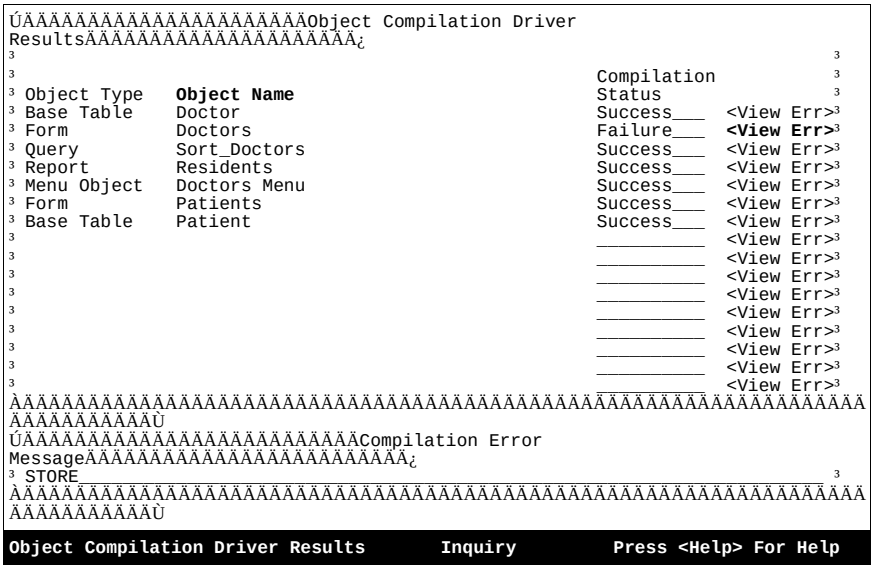
The Object Compile Results window lists all objects in the compilation list by object type and object name and display a message in the Compilation Status column to indicate the results of compilation. This message can be either “Success” or “Failure”.

Compilation Error Messages

For objects whose Compilation Status message is “Failure”, you may view the error message associated with the failure of the compilation.

To do this, use the <UP ARROW> and <DOWN ARROW> keys to position the cursor on the <View Err> field corresponding to the failed object.

The error message displays the Compilation Error Messages window at the bottom of the screen, as shown below:



See your system guide for an explanation of Open M/SQL error messages.

Press the <PREVIOUS> key to return to the Object Compile Driver main window.

Compiling a Compilation Configuration from M Program Code

You can run predefined compilation configurations directly from M program code using the “objcd” entry point to the %mcompil utility.

To do this, use the following syntax:

```
do objcd^%mcompil(Ocd_Identifier)
```

where *Ocd_Identifier* is the name or ID# of a compilation configuration.

Checking the Integrity of Open M/SQL Objects

The Object Integrity Checker utility checks the integrity of various object definitions in the current directory and prepares a report of all integrity errors that it finds. This utility enables application developers to easily identify and repair integrity errors in their applications.

InterSystems recommends that you run the Integrity Checker utility on your applications at a regular interval, such as once a week. You also should run it before exporting objects to a new database.

You can use the Object Integrity Checker to check any of the following objects:

- All Objects
- Base Tables
- Views
- Forms
- Reports
- Menu Objects
- Triggers

It checks all objects of the specified type that reside in the current directory and generates a report of integrity errors. You can send this report to a flat file or printer.

You can set the Object Integrity Checker to automatically correct some of the integrity errors it finds. When it corrects an error, it reports the correction in the Integrity Check Results report. The Integrity Checker cannot fix all errors it finds. When it encounters an error that it cannot fix, it also reports this in the Integrity Check Results report.

When correcting object integrity problems, the Object Integrity Checker does not check privileges on the objects it modifies. It assumes that the Database Administrator is running the utility.

The following sections list the checks made by the Integrity Checker on each supported object.

Checks on Base Tables

For base tables, the Integrity Checker makes sure that:

- The base table has a name
- The field definition for each field in the base table is intact
- The base table does not have any pointers to non-existent triggers
- Any Designative Reference fields in the base table point to a designated table that exists
- All fields that are referenced in the computation code for computed fields are valid
- All fields that are part of the “Computation produces a NULL result IF” code for computed fields are valid
- All fields that are referenced in the “Update field when following field changes” list are valid
- The base table definition stores a list of all views the base table is defined on. The Integrity Checker makes sure all the views in this list actually exist.
- The base table definition stores a list of all forms for which it serves as the data source. The Integrity Checker makes sure all the forms in this list actually exist.
- The “ifn” index is defined for each field in the base table
- All base tables defined in the “m” index exist
- All base tables defined in the “rv” index exist
- All indexed fields defined in the “ifn” index exist
- All indexed fields defined in the “field name” index exist

Checks on Views

For views, the Integrity Checker makes sure that:

- The view’s starting table is valid (it must be a base table or view)
- All table links in the ^mdd(3,ivn,10) tree are valid (Note: These links are not mapped; they are set up in %daview.)
- All fields in the view are valid base table fields.
- All view link tables are valid
- The view field links index (“f1 index”) are valid base table fields
- All fields in the view have field names

Checks on Forms

For forms, the Integrity Checker makes sure that:

- The form's data source is valid (if the form has a data source)
- All database fields exist
- All form-only fields exist
- The form does not have any pointers to non-existent triggers
- All Designative Display fields have valid field names.
- All form-only fields have field names that do not duplicate the field names of database fields on the same form.
- All fields in the window-order list exist (if the form is a window-ordered form)
- All fields that are referenced in the computation code for computed fields are valid
- All fields that are part of the "Computation produces a NULL result IF" code for computed fields are valid
- All fields that are referenced in the "Update field when following field changes" list are valid

Checks on Reports

For reports, the Integrity Checker makes sure that:

- The report's data source is valid (the data source may be base table, view, or query)
- Each report column is a valid field (base table, view, report, or query field)
- Each report sort field is a valid field (base table, view, report, or query field)
- Each sort trailer field is a valid field (base table, view, report, or query field)
- All report summary computed fields are valid fields. Checks the FieldName, Within, and ForEach fields (base table, view, report, or query fields)
- Report data selection conditions reference valid fields Checks FieldName and CompField fields (base table, view, report, or query fields.)
- Report header, report trailer, page header, and page trailer fields are valid fields (base table, view, report, or query fields.)
- All field references in the M code lines for report computed fields are valid.
- If it is a report chain, all members of the chain are valid reports

Checks on Menu Objects

For menu objects, the Integrity Checker utility makes sure that:

- The menu object has a name
- If the menu object is tied to a form, the parent form exists
- The menu object has at least one option

Checks on Menu Object Options

For menu object options, the Integrity Checker utility makes sure that:

- The option text is not null
- The option has an action type
- The option has an action name (if required)

If the menu object option action calls another object, the Integrity Checker makes the following checks:

Table 12-8: Integrity Checks Made When a Menu Option Calls Another Object

If the called object is a ...	Make sure that ...
Form	The form exists
Report	The report exists
Query	The query exists
Pop-Up Menu	The menu object exists
Menu Bar Menu	The menu object exists
Help Topic	The help topic exists
Window	- It is a tied form - The form the menu is tied to is the same as the menu the window is on - The window exists on the form

Checks on Triggers

The Integrity Checker checks all triggers to make sure that:

- All base table trigger pointers exist
- All form trigger pointers exist
- All report trigger pointers exist
- All trigger definitions are valid
- All trigger items are valid

Base Table Trigger Checks

For all base tables, the Integrity Checker checks the validity of:

- All Pre-Filing INSERT, UPDATE, and DELETE triggers
- All Post-Filing INSERT, UPDATE, and DELETE triggers

Form Trigger Checks

For all forms, the Integrity Checker checks the validity of:

- All Pre-Form triggers
- All Post-Retrieval triggers
- All Post-Form triggers
- All Pre-Row triggers (for multi-row forms only)
- All Post-Row triggers (for multi-row forms only)
- All Pre-Window triggers
- All Post-Window triggers
- All Pre-Field triggers
- All Post-Field triggers

Report Trigger Checks

For all reports, the Integrity Checker checks the validity of:

- All Pre-Report triggers
- All Post-Report triggers

Trigger Definition Checks

For all trigger definitions, the Integrity Checker checks to make sure:

- The trigger definition is valid
- Integrity from trigger definitions back to their host objects is valid
- If the trigger calling type is a form, the form exists.
- If the trigger calling type is a base table, the base table exists.
- If the trigger calling type is a report, the report exists

Trigger Items

The Integrity Checker makes the following checks on the validity of form trigger items:

Table 12-9: Trigger Items Verified by the Integrity Checker

If the trigger action type is ...	The Integrity Checker checks the existence of ...
Menu	The menu
Form	The form
Query	The query
Window	The window
Go To Field	The target field
Set Field	The target field
Erase Form	The target form
Erase Window	The target window
Display Window	The target window
Display Form	The target form
Pop-up Menu	The target menu object
Menu Bar	The target menu object

Running the Integrity Checker Utility

When running the Integrity Checker utility, your Open M/SQL database must be perfectly idle. No activity that affects the Open M/SQL objects being checked is allowed to occur.

To ensure that this is the case, the Integrity Checker will not run if it detects that the ^mlock global is defined. The presence of the ^mlock global indicates that Open M/SQL objects are in use.

Al users must exit Open M/SQL before you can run this utility.

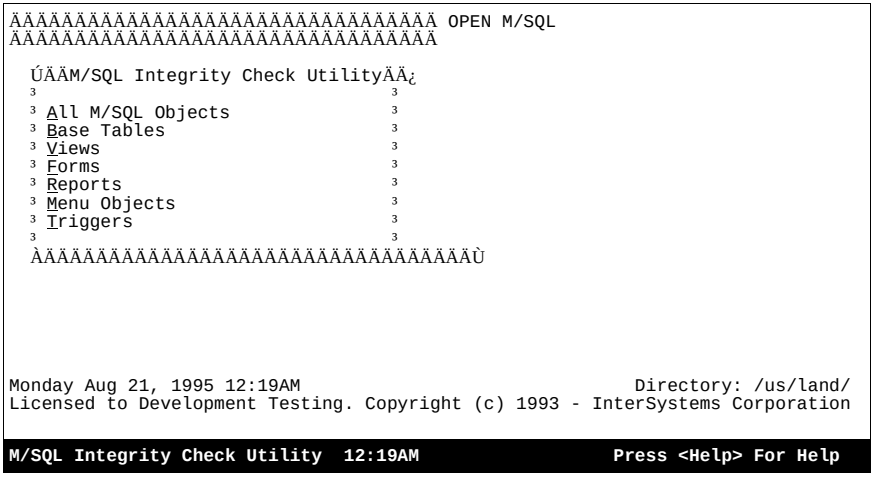
Once the Integrity Checker starts, users may not enter Open M/SQL or run any Open M/SQL objects.

When correcting object integrity problems, the Object Integrity Checker does not check privileges on the objects it modifies. It assumes that the Database Administrator is running the utility.

Procedure **To run the Open M/SQL Integrity Checker utility:**

1. **From the Developer Utilities menu, select the M/SQL Object Integrity Checking option.**

   You see the M/SQL Integrity Check Utility menu, as shown below:



Note: If your Open M/SQL database is not completely idle, you see the following message:

```
The ^mlock global is defined in this database
which indicates OPEN M/SQL is in use. Please run
the OPEN M/SQL integrity checkers when the OPEN
M/SQL objects are not in use in this database.
```

You cannot run the Integrity Checker utility while there are users in Open M/SQL.

2. **From the M/SQL Integrity Check Utility menu, select the appropriate option for the objects you want to check.**

   If you selected "Trigger Definitions", you see following window:

```
     THIS UTILITY WILL CHECK FOR INCONSISTENCIES IN ALL OPEN M/SQL TRIGGER
     DEFINITIONS. YOU HAVE THE OPTION OF JUST PRINTING THE ERRORS, OR
     HAVING OPEN M/SQL FIX THE PROBLEMS IT CAN AUTOMATICALLY.

Do You Want The Integrity Checker To Fix Errors?
```

3. **Answer *Yes* or *No* at the prompt "Do You Want The Integrity Checker To Fix Errors?" to indicate whether or not you want the Integrity Checker to correct some of the integrity errors it finds.**

   Note: You may type *Q* or ^ to exit the Integrity Checker utility.

   The Object Integrity Checker can correct some but not all of the integrity errors it finds. When it corrects an error, it reports the correction in the Integrity Check Results report. When it encounters an error that it cannot fix, it also reports this in the Integrity Check Results report. You must correct these errors manually.

If you answer *Yes* at the "Do You Want The Integrity Checker To Fix Errors?" prompt, you see the following prompt asking you to specify which error fixing mode you want the Integrity Checker to use:

```
Select Error Fixing Mode:
   1 - Fix all errors without prompting
   2 - Prompt for permission to fix each error
Select Error Fixing Mode:
```

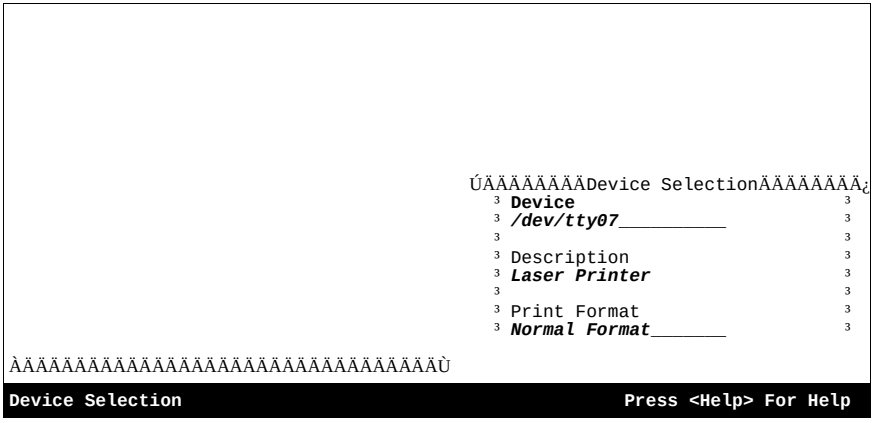
Type *1* to specify "Fix without prompting" mode, where the Integrity Checker automatically fixes any errors that it can fix.

Type *2* to specify "Prompt before fixing" mode, where the Integrity Checker displays each error, indicates whether or not it can be fixed automatically, and prompts the user with to "confirm" before fixing it.

Note: When correcting object integrity problems, the Object Integrity Checker does not check privileges on the objects it modifies. It assumes that the Database Administrator is running the utility.

If you answer *No* at the "Do You Want The Integrity Checker To Fix Errors?" prompt, the Integrity Checker will not correct any errors that it finds, though it still reports all errors in the Integrity Check Results report.

You see the Device Selection window, as shown below:



The following table lists and describes the fields located on the Device Selection window:

Table 12-10: Fields on Device Selection Window

Field	Description
Device	This field always defaults to your current device (the current value of $IO). You may change the default to any valid output device to which your device is linked.
Description	This field reflects the description given to the specified device in the device table.
Print Format	Here you may select any print format defined for the specified device.

4. **In the Device field, enter the name of the device to which you want to send this report.**

   You can send the report to any valid output device that is linked to your current device. The default device is your current device.

   To send the report to your screen, press the <PROCEED> key.

   To send the report to another device (such as a printer), delete the name of the default device at the Device prompt, enter the name of the new target device, adjust the Print Format parameter as appropriate, and press the <PROCEED> key.

5. **When you have entered a device name and the appropriate print format parameters, press the <PROCEED> key.**

   This executes the Integrity Check Results report.

   Below is a sample Integrity Check Results report for Trigger Definitions:

```
TYP  OBJ NAME              TRIG# ITEM ERROR
---  -------------------- ----- ---- ----------------------------------------
frm  Doctors          ...  1     1    FORM Trigger calls an undefined form.
                                       Trigger Location - PostFld Trig: field3
                                       This must be fixed manually.

frm  Big Test         ...  166   1    POPUP MENU Trigger calls an undefined
                                       menu object.  Trigger Location - PostFld
                                       Trig: BigPopUpTest  This must be fixed
                                       manually.

???  ReportTriggers        225        Pre-Report Trigger not called from
                                       report.  Trigger Location - Pre-Report

???  ReportTriggers        226        Post-Report Trigger not called from
                                       report.  Trigger Location - Post-Report

frm  JLD237           ...  289   1    FORM Trigger calls an undefined form.
                                       Trigger Location - PostFld Trig:
                                       GlucoseTest  This must be fixed manually.


MORE ('^' or 'Q' to quit) >
```

Note: When output to the screen, the report displays one screen of errors at a time and prompts you to press <RETURN> to scroll ahead to the next screen.

The Integrity Check Results report for Trigger Definitions provides the following information about each integrity error it finds:

- Object Type — type of object to which the trigger belongs. It may be a form (frm), a base Table (tbl), or a Report (rpt). Sometimes, the object to which the trigger belongs is not know (???)
- Object Name — name of the specific object to which the trigger belongs (if the object is known)
- Trigger # — internal identification number of the trigger in which the integrity error was found
- Trigger Item # — number of the particular item in the trigger item sequence in which the integrity error was found
- Error Message — message that describes the integrity error

6. **Press the <PREVIOUS> key to exit the report.**

Error Fixing Mode

If you have enabled error fixing, the Integrity Checker can automatically correct some of the errors it finds.

Error fixing has two modes:

1. Fix errors without prompting
2. Prompt before fixing errors

The behavior of the Integrity Checker depends on which of these modes you select.

Fix Errors Without Prompting

In this mode, the Integrity Checker displays one error per screen and automatically fixes all errors that it can fix.

If it fixes the error, you see the message:

```
Fixed
```

If it cannot fix the error, you see the message:

```
This error must be manually fixed
```

Prompt Before Fixing Errors

In this mode, the Integrity Checker displays one error per screen and prompts the user to confirm a fix before it fixes an error.

If it can fix the error, you see the message:

```
OK to fix this error?
```

If it cannot fix the error, you see the message:

```
This error must be manually fixed
```

Searching for Strings in Open M/SQL Objects

The Object String Search utility searches through Open M/SQL object definitions for specified text strings.

The Object String Search utility can search any of the following object types:

- Base Tables
- Views
- Queries
- Reports
- Forms
- Menus (Old Style)
- Menu Objects
- Help Topics
- Help Documents

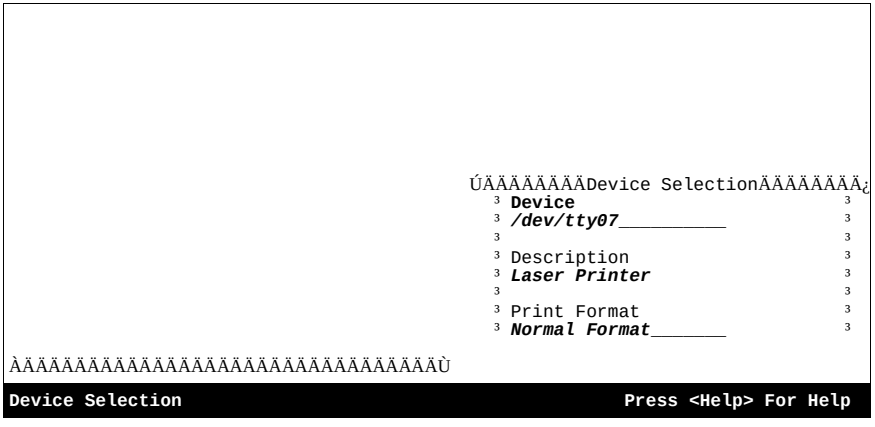
The utility can search through all objects in the current directory, or it can search through selected objects. You may select objects by object type, and you may further select objects by specific object name within a certain object type.

The utility can search for one string or many different strings.

Procedure **To use the Object String Search utility:**

1. **From the Developer Utilities menu, select the M/SQL Object Integrity Checking option.**

   You see the Device Selection window, as shown below:



2. **In the Device field, enter the name of the device to which you want to send this report.**

   You can send the report to any valid output device that is linked to your current device. The default device is your current device.

   To send the report to your screen, press the <PROCEED> key.

   To send the report to another device (such as a printer), delete the name of the default device at the Device prompt, enter the name of the new target device, adjust the Print Format parameter as appropriate, and press the <PROCEED> key.

3. **When you have entered a device name and the appropriate print format parameters, press the <PROCEED> key.**

4. **You see the following prompt:**

   ```
   Search String #1 (^to quit):
   ```

   Here, you may enter the text of the first search string, and press <RETURN>.

   You may enter any number of search strings.

   Note: The list of search strings is an inclusive OR list, which means that if the utility finds ANY one of the specified search strings within an object definition, it returns a "string found" message.

   Each time you enter a new search string, you see the following prompt:

   ```
   EXACT MATCH? No==>
   ```

5. **At the EXACT MATCH prompt, press <RETURN> to accept the *No* default, or type *Y* and press <RETURN>.**

   Answer *Yes* to indicate that you want the Object String Search utility to search for an EXACT match (case sensitive) of the specified string.

   Answer *No* to indicate that you want the Object String Search utility to search for an ALPHAUP match (case insensitive) of the specified string.

   The utility continues to prompt you to enter additional search strings.

6. **When you have finished entering search strings, press <RETURN> at the Search String prompt.**

   You see the following prompt:

   ```
   Object Type To Search (^to quit, ? for list):
   ```

7. **At the Object Type To Search prompt, you may specify the object type(s) you want to search.**

To search all object types, enter *. Otherwise, enter a number from 1 to 9 to represent the object type you want to search. Type *?* to see a list of the object type choices and their numbers. The table below shows the number that corresponds to each object type:

Object Type	#
Base Tables	1
Views	2
Queries	3
Reports	4
Forms	5
Menus (old-style)	6
Menu Objects	7
Help Topics	8
Help Documents	9
All object types	*

You may specify one object type or all object types.

When you select an object type, you see the following prompt:

```
<Object Type> (enter name or * for all)
```

8. **At the Object Name prompt, you may enter the names of the specific objects you want the utility to search.**

   For example, if you specified Base Tables as the object type, you may enter any number of specific base table names. Or, to search through all objects of the specified type, you may type * at this prompt.

   When selecting object names, you may type the first few characters of an object name, then press the <SEARCH CURRENT TABLE> key to see a lookup box that lists all matching entries for the specified object type.

   Note: If you are searching through all object types, the utility automatically searches through all object names within each object type. You cannot restrict this search.

9. **When you have finished entering object names, press <RETURN> at the Object Name prompt.**

   The utility sends output to the specified device.

   The Object String Search utility outputs a header page that lists each search string and all of its criteria, including the string matching criteria ([EXACT] or [ANY]).

   A sample of the Open M/SQL Object Text Search header page is shown below:

```
                    OPEN M/SQL OBJECT TEXT SEARCH PRINTOUT

Search Criteria:
 1) [ ANY ]   leo
<PRESS RETURN>
```

10. **Press <RETURN> to view the String Search report.**

    Following the header page, the utility outputs a detailed listing of all matches found. For each match, the report provides the following information:

    - Object Type
    - Object Name
    - Sub-object Type (when applicable)
    - Sub-object Name (when applicable)
    - Location (attribute) where the match was found
    - Text string in which the match was found

For example, you might see the following entry for an EXACT match on string "^ABC" found in the INSERT Validation Code for Base Table "Patients":

```
Base Table - 'Patients'
LOCATION: Insert Validation Code
TEXT:     i {Status}="" s %ok=0 d ^ABC({Patient})
```

Typically, the report identifies the exact location of the matching text string. For example, if it finds a match within Additional Validation Code for a field (which can have any number of text lines defined), the report prints only the matching text lines. In the case of individual trigger actions and individual lookup specifications, the report prints the entire trigger action specification or entire lookup specification, in order to provide a meaningful context for the text match.

Below is a sample of the String Search report:

```
Base Table: 'jld087'
LOCATION = Base Table Name
TEXT     = jld087

Base Table: 'jld087'
LOCATION = Base Table Description
TEXT     = jld087 Test               (irn = 1)

Base Table: 'jld087' - Field: 'jld087'
LOCATION = Field Name
TEXT     = jld087

Base Table: 'jld087' - Field: 'jld087'
LOCATION = Field Description
TEXT     = jld087 Row ID

Base Table: 'jld087' - Map: 'Data Master Map' - Access Level: L3
LOCATION = Map Access Level Expression
TEXT     = {jld087}

MORE ('^' or 'Q' to quit) >
```

11. Press the <PREVIOUS> key to exit the report.

Invoking Macro Source Routine Utilities

Open M/SQL provides a number of utilities for examining and manipulating M macro source code routines and include files. These utilities are useful for both developing and maintaining Open M/SQL applications.

The Open M/SQL routine utilities include:

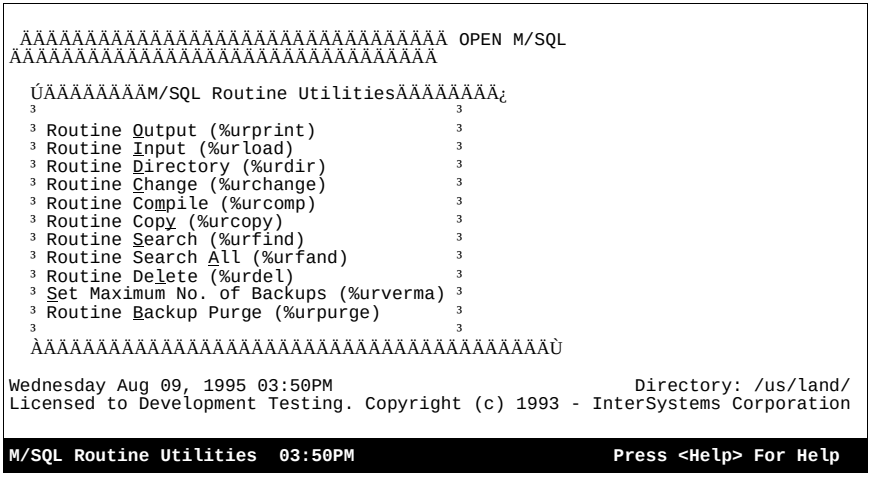
- Routine Output (%urprint)
- Routine Input (%urload)
- Routine Directory (%urdir)
- Routine Change (%urchange)
- Routine Compile (%urcomp)
- Routine Copy (%urcopy)
- Routine Search (%urfind)
- Routine Search All (%urfand)
- Routine Delete (%urdel)
- Set Maximum Number of Backups (%urverma)
- Routine Backup Purge (%urpurge)

You may invoke and use these utilities via the Developer Utilities menu.

Procedure **To run macro source routine utilities from the Developer Utilities menu:**

1. **From the Developer Utilities menu, select the Macro Routine Utilities option.**

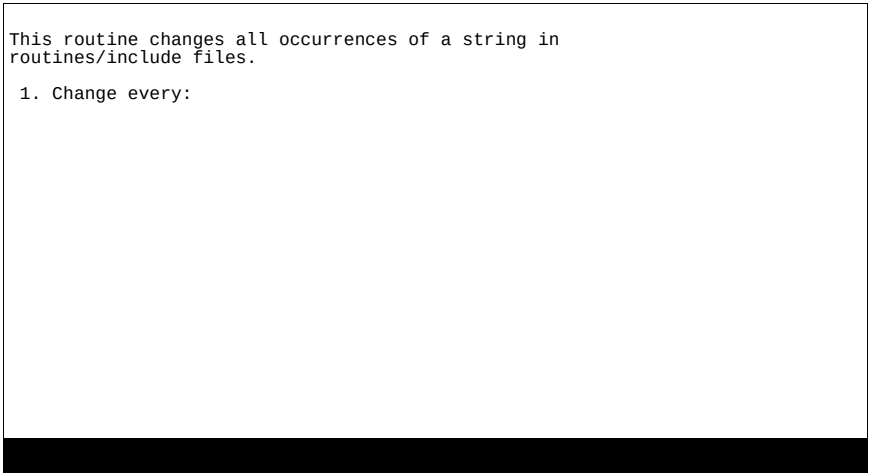
   You see the M/SQL Routine Utilities menu, as shown below:



2. **From the M/SQL Routine Utilities menu, select the routine that you want to run.**

   Open M/SQL runs the selected routine utility in a window.

   For example, if you selected the Routine Search utility (%urfind), you would see the following screen:



3. **Answer all prompts as appropriate.**

   Note: For complete information on how to use all of the Open M/SQL routine utilities, see Chapter 7, *Open M/SQL Routine Management Utilities*.

   When you exit the selected routine utility, you return to the Routine Utilities menu.

Querying Objects by Routine Prefix

The Object Routine Prefix utility lets you look up any routine prefix that is associated with an Open M/SQL object in the current directory.

When you look up a routine prefix, the utility displays information about where that routine is being used. Specifically, it provides the following information:

- Routine Prefix
- Object Type
- Object Identification #
- Object Name

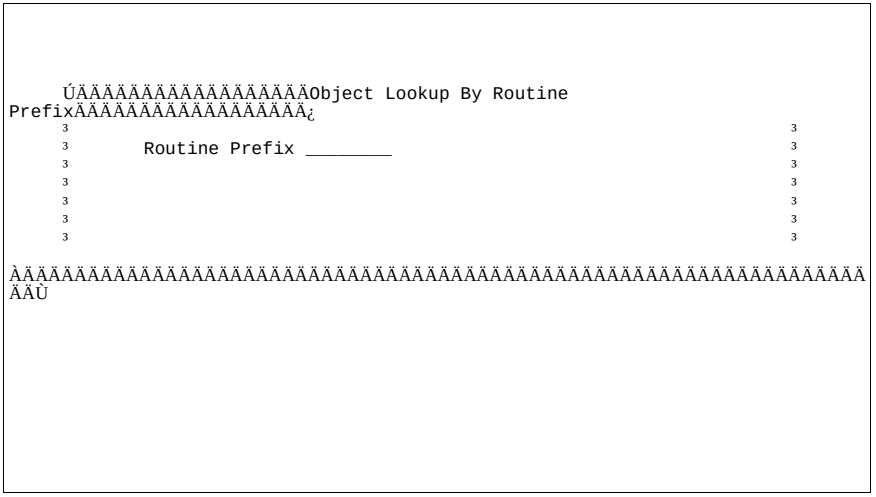
When compiling large applications, you may receive warning and error messages that cite routine names. These messages do not always specify which object the routine belongs to. For example, the message “Syntax error in mp3261” does not provide any information about routine “mrp3261”. In order to correct this error, you need to find out where this routine is located. The Object Routine Prefix utility enables you to do this.

This utility is also useful when deleting routines to clean up your directory

Procedure **To use the Object Routine Prefix utility:**

1. **From the Developer Utilities menu, select the Query Object By Routine Prefix option.**

   You see the Object Lookup by Routine Prefix lookup window, as shown below:



2. **At the Routine Prefix field, enter the name of the routine prefix you want to look up.**

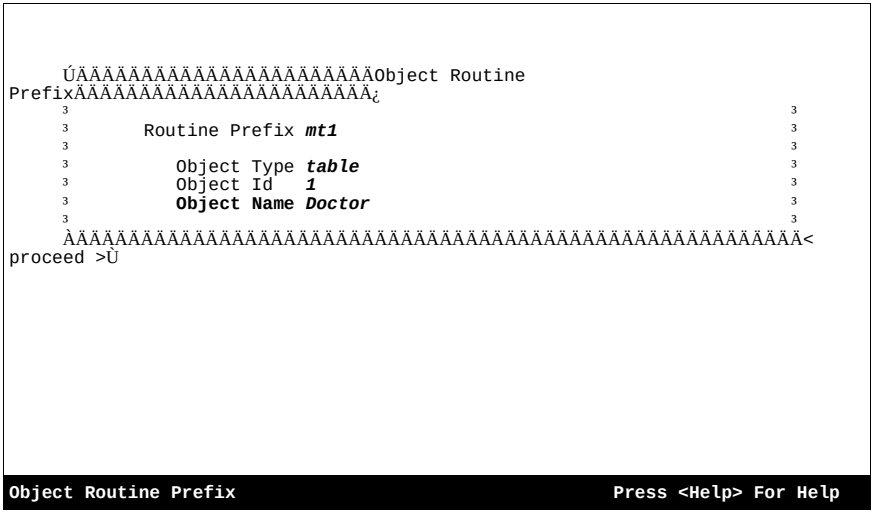
   You may perform both unqualified and qualified lookups on the directory of routine prefix names.

   To perform an unqualified lookup, press the <SEARCH CURRENT TABLE> key with the Routine Prefix field blank. You see a lookup box that lists all routine prefixes.

   To perform a qualified lookup, enter the first one or more characters of the routine prefix you want to find, and press the <SEARCH CURRENT TABLE> key. You see a lookup box that lists all matching entries.

3. **Within the lookup box, use the <UP ARROW> and <DOWN ARROW> keys to position the cursor on the routine prefix you want to examine, and press <RETURN> to select it.**

   You see the Object Routine Prefix window, as shown below:



   The Object Routine Prefix window displays information about the routine prefix you selected.

   All fields on this window are read-only.

4. **To exit the Object Routine Prefix window, press <RETURN>.**

You return to an empty Object Lookup By Routine Prefix window, where you may select another routine prefix to examine.

National Language Reports

The National Language Reports utility lets you generate a screen or hard copy report that displays the translations provided for all system-generated messages, key labels, menu titles, and menu option text throughout the Open M/SQL environment.

The report is organized alphabetically by message. For each message, the report provides the following information:

- The English text for the message, key label, menu title, or menu option text
- The maximum translation length (based on the length of the English text)
- List of supported languages with the appropriate translation of the message (if supplied)

Procedure **To print the National Language report:**

1. **From the Developer Utilities menu, select the National Language Reports option.**

   You see the Device Selection window.

2. **In the Device field, enter the name of the device to which you want to send this report.**

   You can send the report to any valid output device that is linked to your current device. The default device is your current device.

   To send the report to your screen, press the <PROCEED> key.

   To send the report to another device (such as a printer), delete the name of the default device at the Device prompt, enter the name of the new target device, adjust the Print Format parameter as appropriate, and press the <PROCEED> key.

3. **When you have entered a device name and the appropriate print format parameters, press the <PROCEED> key.**

   This executes the National Language report.

Below is a sample page from the National Language Report:

```
Enter Translations For System MPage: 3

WARNING:  Ensure that length of translations for system messages does not
          exceed the length of the English text being translated.  Call
          InterSystems if this limitation is unacceptable.

          Also, if no English text is provided, it is not necessary
          to enter translations.

MESSAGE CONTEXT: ALTER
English Message: %ALTER
Maximum Length:  15

Enter Translations:
     Dansk (Danish)       -
     Deutsch (German)     - %ALTER
     Dutch (Netherlands)  -
     Espanol (Spanish)    -
     Finska (Finnish)     -
     Francais (French)    - %ALTER
     Italiano (Italian)   - %Altera
     Norsk (Norwegian)    - %Forandere
     Portuguese           - Altere
     Svenska (Swedish)    - %Andra
Press <Return> to continue, <Options> to scroll, <Exit> to Exit
```

Note: When output to the screen, this report displays one screen at a time and prompts you to press <RETURN> to scroll ahead to the next screen.

4. **You may press the <PREVIOUS> key at any time to exit the National Language Report and return to the Developer Utilities menu.**

APPENDIX

A

SQL Error Messages

Whenever you issue an SQL statement, it returns a non-descriptive numerical message to the variable SQLCODE.

The message indicates one of the following:

- Successful completion
- Error

The tables in this Appendix list all message codes that can be returned to the SQLCODE variable and describes the meaning of each code.

Successful Completion Messages

The following table lists the codes for successful completion:

Table A-1: Codes Returned on Successful Completion

Error Code Number	Meaning
0	Query completed successfully
100	Query completed successfully, but no row was found to satisfy the query

Error Messages

The following table lists the codes for errors found during parsing:

Table A-2: SQL Error Messages

Error Code Number	Function
-1	Invalid SQL statement
-2	Exponent digits missing after 'E'
-3	Closing quote (") missing
-4	A term expected, beginning with either of: identifier, constant, aggregate, %ALPHAUP,%UPPER, %EXACT, $$, :, +, -, (, NOT, or EXISTS
-5	Column-number specified in ORDER does not match the SELECT list
-6	ORDER must specify column number, not names, when after UNION
-7	ORDER column is not in SELECT list
-9	Incompatible SELECT lists used in UNION
-10	The SELECT list of the query must have exactly one item
-11	A scalar expression expected, not a condition
-12	A term expected, beginning with either of: identifier, constant, aggregate, $$, (, :, +, -, or %ALPHAUP, %UPPER, or %EXACT
-13	An expression other than a subquery expected here
-14	A comparison operator is required here
-15	A condition expected after NOT
-16	A qualifier SOME or ALL expected after the FOR in the for-expression
-17	A for-condition expected after the (in the for-expression
-18	IS (IS NOT) NULL predicate can be applied only to a field
-19	An aggregate function cannot be used in a WHERE clause
-20	Name conflict in the FROM list over label <label>
-22	ORDER must specify column names, not numbers, when after 'SELECT *'
-23	Label <label> is not listed in FROM
-25	Input (<token>) encountered after end of query
-26	Missing FROM clause
-27	Ambiguous labels for field <name>
-28	Host variable name must begin with either % or a letter, not <symbol>
-30	Table <tablename> not found

Table A-2: SQL Error Messages (Continued)

Error Code Number	Function
-31	Field <fieldname> not found in table(s) <tablename>
-32	Outer join symbol (=*) must be between two fields
-33	No fields found for table <tablename>
-34	Contradictory conditions: 'f is NULL' vs. 'f=constant'
-35	Contradictory conditions: 'f IS NULL' vs. 'f in range'
-36	Contradictory conditions: 'f IS NULL' vs. 'f=expression'
-37	Contradictory conditions: constants should satisfy <condition>
-38	No master map for table <tablename>
39	No Row ID field for table <tablename>
-41	An extrinsic function call must have the form $$tag^rou(...
-42	Closing quotes (") missing following pattern match
-51	An SQL statement expected, <token> found
-52	Cursor <cursorname> already DECLAREd or not DECLAREd
-53	Constant or variable expected as new value
-54	Array designator (last subscript omitted) expected after VALUES
-55	Invalid GRANT <role> to or revoke <role> from
-56	Action not applicable to an object of this type
-58	Object type not found
-59	Cannot have more than one field
-60	An action (%ALTEr, SELECT, UPDATE, etc.) expected
-61	Cursor not updatable
-62	Additional new values expected for INSERT/UPDATE
-63	Data exception - invalid escape character
-70	Incorrectly defined RowID
-99	Privilege violation
-101	OPEN attempted on a cursor that is already open
-102	Cursor operation (FETCH, CLOSE, UPDATE, DELETE) attempted on an unopened cursor
-103	Positioned UPDATE or DELETE attempted but the cursor is not positioned on any row
-104	Field validation failed in INSERT

Table A-2: SQL Error Messages (Continued)

Error Code Number	Function
-105	Field validation failed in UPDATE
-106	Row to DELETE not found
-107	Table validation failed
-108	Required field missing; INSERT or UPDATE not allowed
-109	Cannot find the row designated for UPDATE
-110	Locking conflict in filing
-111	Operation not licensed on this system
-112	Access violation
-113	%THRESHOLD violation
-201	Table or view name not unique
-212	Unrecognized or invalid input device
-213	Keyword ON expected
-214	Left parenthesis expected
-215	Right parenthesis expected
-216	Comma expected
-217	Unexpected End-of-Field
-218	Unexpected End-of-Query
-219	Incorrect or missing field name
-220	Incorrect or missing data type
-221	Invalid length/scale specification
-222	Default value exceed field length
-223	Datatype/qualifier mismatch
-224	Unrecognized or conflicting qualifiers
-225	Multiple primary key specification
-226	Incomplete primary fields specification
-227	Missing quote introducing or ending description
-228	Missing or illegal global name
-229	Query rejected
-230	Bad column constraint

Table A-2: SQL Error Messages (Continued)

Error Code Number	Function
-231	SQL reserved word not allowed **Note**: For a list of Open M/SQL reserved words, see Appendix B.
-232	Single quotation expected instead of double quotation
-233	Repeated key definition or conflict constraints
-234	Bad default value
-235	Repeated index definition
-236	Data type error
-237	More than one index defined at the same time here
-238	More than one CREATE TABLE statement in one SQL macro
-239	More than one DROP TABLE statement in one SQL macro
-240	The field is required and can't have NULL value
-242	General DDL parsing error
-243	Scales with character data types
-244	Scales with integer data types
-245	Specifying length for this data type not allowed
-246	Decimal part with integer default value
-248	Default value is not allowed for this data type
-1000	Maximum open cursors exceeded **Note**: Up to 255 cursors are permitted for a given connection
-1001	Invalid cursor
-1002	Fetch out of sequence
-1003	No statement parsed
-1006	Bind variable does not exist
-1007	Variable not in select list
-1008	Not all variables bound
-1012	Not logged on
-1013	User requested cancellation of current operation
-1017	Invalid user name/password
-1024	Invalid data type in 'obndrv', 'obndrn', or 'odefin' call
-1031	Insufficient privileges
-1405	Warning: Fetched column value is NULL

Table A-2: SQL Error Messages (Continued)

Error Code Number	Function
-1406	Warning: Fetched column value was truncated
-1454	Cannot convert select item value to numeric data type
-1455	Value overflows specified integer data type
-1459	Invalid length specified for variable character string
-1483	Invalid length for DATE or NUMBER bind variable

Note Error codes used by the Open M/SQL Relational Client and Server are documented in the *Open M/SQL Relational Client User Guide*.

APPENDIX

B

Open M/SQL Reserved Words

Open M/SQL reserves certain keywords for its own use.

This means that you cannot use these words as identifiers (names of database objects such as base tables, forms, fields, etc.).

The following table lists the Open M/SQL reserved words:

Table B-1: Open M/SQL Reserved Words

%AFTERHAVING	%ALPHAUP	%ALTER
%BEGTRANS	%CATALOG	%CHECKPRIV
%EXACT	%EXTERNAL	%FOREACH
%FORM	%INTALL	%INTERNAL
%INTOBUILD	%INTRANS	%LEVEL
%MENU	%MENUOBJ	%NOCHECK
%QUERY	%REPORT	%ROWCOUNT
%STARTSWITH	%THRESHOLD	%UPPER
ADMIN	ALL	ALTER
AND	ANY	AS
ASC	AUTHORIZATION	AVG
BEGIN	BETWEEN	BY
CASCADE	CHAR	CHARACTER
CHECK	CLOSE	COBOL
COMMIT	CONSTRAINT	CONTINUE
COUNT	CREATE	CURRENT
CURSOR	DATE	DEC

Table B-1: Open M/SQL Reserved Words (Continued)

DECIMAL	DECLARE	DELETE
DESC	DISTINCT	DOUBLE
DROP	END	ESCAPE
EXEC	EXISTS	FETCH
FILE	FLOAT	FOR
FOREIGN	FORTRAN	FOUND
FROM	GO	GOTO
GRANT	GROUP	HAVING
IN	INDEX	INDICATOR
INSERT	INT	INTEGER
INTO	IS	LANGUAGE
LIKE	MAX	MIN
MODULE	NOT	NULL
NUMERIC	OF	ON
OPEN	OPTION	OR
ORDER	PASCAL	PLI
PRECISION	PRIMARY	PRIVILEGES
PROCEDURE	PUBLIC	REAL
REFERENCES	RELATION	REVOKE
ROLE	ROLLBACK	SCHEMA
SECTION	SELECT	SET
SMALLINT	SOME	SQL
SQLCODE	SQLERROR	SOME
TABLE	TIME	TO
UNION	UNIQUE	UPDATE
USER	VALUES	VARCHAR
VARYING	VIEW	WHENEVER‘
WHERE	WITH	WORK

Note Some of these reserved words are used for the SQL Data Definition Language (DDL), which is currently under implementation by InterSystems.

APPENDIX

C

Open M/SQL Supported Terminal Types

Open M/SQL supports the following terminal types:

Table C-1: Open M/SQL Supported Terminal Types

Type	Name	Description
Video	ALTOS 5	Altos 5 Terminal
Video	ANSI	ANSI Terminal
Video	ANSI Color	ANSI Color Terminal
Video	CIT-500	CIT-500 (Emulates VT 132)
Video	COBRA	COBRA Terminal
Video	D200 General	Dasher Terminal D214 & D215
Video	D200 Gen/Perfect	Dasher Terminal D214 & D215
Video	D470C	Dasher Terminal DG mode
Video	DTM PC (Supported for Open M/SQL on DTM)	DTM PC Console
Video	DTM PC Color	DTM PC Color Console
Video	FALCO	Falco Emulating a VT 220
Video	Generic	Pure Roll-and-Scroll
Video	IBM 3151	IBM 3151 ASCII Display Station
Video	IBM 6091	IBM 6091-19 Terminal
Video	LANSI	Long ANSI Terminal
Video	MSM PC (Supported for Open M/SQL on MSM)	MSM PC Console
Video	MSM PC Color	MSM PC Color Console

Table C-1: Open M/SQL Supported Terminal Types (Continued)

Type	Name	Description
Video	PC	IBM PC Terminal
Video	PC Color	IBM PC Terminal With Wired Color
Video	QUME	QUME Terminal
Video	SUN	SUN Terminal
Video	TV9320	TeleVideo 9320 Terminal
Video	VT100 (Supported for Open M/SQL on DSM & MSM)	DEC VT 100 Terminal
Video	VT220 (Supported for Open M/SQL on DSM & MSM)	DEC VT 220 Terminal
Video	VT240 (Supported for Open M/SQL on DSM & MSM)	DEC VT 240 Terminal (Regis-md)
Video	VT420 (Supported for Open M/SQL on DSM & MSM)	DEC VT 420 Terminal
Video	WYSE30	WYSE30
Video	WYSE60 (D210)	WYSE60 (Emulating Dasher D210)
Video	WYSE60N	WYSE60 Native Mode Terminal
Video	WYSE85	C-WYSE-85 (DEC VT 220 Emulation)
Hard Copy Video	DEC	
Printer	HPIII	Hewlett-Packard LaserJet III
Printer	LASER	Generic Laser Printer
Printer	LN03	Digital LN03
Printer	LT100	DEC Letterprinter 100
Printer	SQ	Sunquest PostScript Printer
Printer	TI	Texas Instruments Printer

Open M/SQL for DSM

Open M/SQL supports the following terminal types for use in the DSM environment:

- DEC VT 100 and compatibles
- DEC VT 220 and compatibles

Open M/SQL for DTM

Open M/SQL supports the DTM-PC Console terminal type for use in the DTM environment. The Console device should always use this terminal type.

Open M/SQL also supports dumb terminals in the DTM environment. If you invoke DTM in multi-user mode, you can log on to a dumb terminal. The default terminal type for a dumb terminal is “VT100”, though you may select any supported terminal type at the “Terminal Type” prompt.

Open M/SQL for MSM Environment

Open M/SQL supports the following terminal types for use in the MSM environment:

- DEC VT 100 and compatibles
- DEC VT 220 and compatibles
- DEC VT 240 and compatibles
- DEC VT 420 and compatibles
- FALCO
- GENERIC
- MSM PC Console
- WYSE85

Open M/SQL supports the MSM PC Console terminal type for use in the Open M/SQL MSM-PC/386 environment. In order to connect terminals other than the MSM PC console, you must use a serial port, an Arnet or other board, or a LAT. The MSM-PC/386 environment supports only keyboards whose function keys generate the DEC VT escape sequences.

It may be possible to use personal computers connected to a serial port, provided that they use VT terminal emulation software that supports the function keys and generates the correct escape sequences.

Some dumb terminals may also work if they support a VT100 or VT220 emulation mode. You should consult with individual hardware vendors to ascertain this.

APPENDIX

D

Full Screen Editor Keyboard Actions

The following tables show the key mapping scheme for using the Full Screen Editor on the keyboards associated with the terminal types supported by Open M/SQL.

The tables below contain the following information:

- The first column, “Editor Action,” lists the actions supported by the Full Screen Editor environment.
- The second column, “Key(s) To Press,” lists the primary keystroke or keystroke sequence used to invoke each action.

Altos

Table D-1: Altos Keyboard Mapping for the Full Screen Editor

Editor Action	Key(s) To Press
Help Menu	<F13><F15>
Beginning of Area	<F13><Left Arrow>
Bottom	<F13><B> <F13><SCRN Next>
Bottom of Window	<F13><Down Arrow>
Buffer Options	<CTRL-E><B> <CTRL-E><CTRL-B>
Copy	<CTRL-E><D> <CTRL-E><CTRL-D>
Cut or Paste	<CTRL-E><C> <CTRL-E><CTRL-C>
Delete Back Character	<DEL>

Table D-1: Altos Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Delete Character	<CTRL-D> <DEL Char>
Delete Word	<CTRL-W>
Do Editor Action	<CTRL-\> <CTRL-E><A> <CTRL-E><\>
Down Arrow	<CTRL-J> <Down Arrow>
End of Area	<F13><CTRL-K> <F13><Right Arrow>
Enhance	<CTRL-G> <F13>
Erase Area	<CTRL-L> <F13><Del>
Find	<CTRL-E><F> <CTRL-E><CTRL-F>
General Help	<CTRL-Y> <CTRL-Z> <F13><Y> <F13><Z> <CTRL-E><?>
Goto Tag	<CTRL-E><G> <CTRL-E><CTRL-G>
Enlarge Current Window	<CTRL-E><+>
Field Help	<F14>
Last Buffer	<CTRL-E><L> <CTRL-E><CTRL-L>
Left Arrow	<Left Arrow> <CTRL-H>
Make Buffer	<CTRL-E><=> <CTRL-E><[>
Mark Options	<CTRL-E><M> <CTRL-E><CTRL-M>
Next Find	<CTRL-E><N> <CTRL-E><CTRL-N>
Next Screen	<SCRN NEXT>
Next Word	<CTRL-F>

Table D-1: Altos Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
One Window	<CTRL-E><1>
Other Options	<CTRL-E><O> <CTRL-E><CTRL-O>
Previous	<F16>
Previous Find	<CTRL-E><P> <CTRL-E><CTRL-P>
Previous Screen	<SCRN Prev>
Previous Word	<CTRL-B>
Query Replace	<CTRL-E><Q> <CTRL-E><R> <CTRL-E><CTRL-Q> <CTRL-E><CTRL-R>
Redraw Screen	<CTRL-E></>
Return	<RETN>
Right Arrow	<CTRL-K> <Right Arrow>
Save and Compile Buffer	<F12> <CTRL-E><F12>
Show Current Time	<CTRL-E><T> <CTRL-E><CTRL-T>
Switch To Other Window	<CTRL-E><W> <CTRL-E><CTRL-W>
Tab	<Tab>
Toggle Select	<CTRL-E><S> <CTRL-E><CTRL-S>
Top	<F13><SCRN Prev> <F13><T>
Top of Window	<F13><Up Arrow>
Two Windows	<CTRL-E><2>
Up Arrow	<Up Arrow>
Use Buffer	<CTRL-E><U> <CTRL-E><CTRL-U>
Window Options	<F15>
Word Capitalize	<CTRL-E><->

Table D-1: Altos Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Word Lowercase	<CTRL-E><_>
Word Uppercase	<CTRL-E><^>

ANSI

Table D-2: ANSI Keyboard Mapping for the Full Screen Editor

Editor Action	Keys(s) To Press
Help Menu	<F1><F3>
Advance	<CTRL-A> <F1><A>
Beginning of Area	<F1><Left Arrow>
Bottom	<CTRL-V> <F1><Page Down> <F1><V>
Bottom of Window	<F1><CTRL-J> <F1><Down Arrow>
Break	<CTRL-C>
Buffer Options	<CTRL-E><B> <CTRL-E><CTRL-B>
Copy	<CTRL-E><D> <CTRL-E><CTRL-D>
Cut or Paste	<CTRL-E><C> <CTRL-E><CTRL-C>
Delete Back Character	< <- > <Delete>
Delete Character	<CTRL-D> <F1><D>
Delete Word	<CTRL-W> <F1-W>

Table D-2: ANSI Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Keys(s) To Press
Do Editor Action	<CTRL-^> <CTRL-E><A> <CTRL-E><CTRL-A>
Down Arrow	<CTRL-J> <Down Arrow>
End of Area	<F1><CTRL-K> <F1><Right Arrow>
Enhance	<CTRL-G> <F1>
Erase Area	<CTRL-L> <F1>< <- >
Erase to Beginning	<F1><Delete Key>
Find	<CTRL-E><F> <CTRL-E><CTRL-F>
General Help	<CTRL-Y> <CTRL-Z> <F1><Y> <F1><Z> <CTRL-E><?>
Goto Tag	<CTRL-E><G> <CTRL-E><CTRL-G>
Enlarge Current Window	<CTRL-E><+>
Field Help	<F2>
Last Buffer	<CTRL-E><L> <CTRL-E><CTRL-L>
Left Arrow	<Left Arrow>
Macro Definition	<F1><K>
Make Buffer	<CTRL-E><=> <CTRL-E><[>
Mark Options	<CTRL-E><M> <CTRL-E><CTRL-M>
Next Error in Buffer	<CTRL-E><E> <CTRL-E><CTRL-E>
Next Find	<F6> <CTRL-E><N> <CTRL-E><CTRL-N>
Next Tag	<CTRL-N> <F1><N>

Table D-2: ANSI Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Keys(s) To Press
Next Screen	<Page Down>
Next Word	<CTRL-F> <F1><F>
One Window	<CTRL-E><1>
Other Options	<CTRL-E><O>
Paste	<Insert>
Previous Find	<CTRL-E><P> <CTRL-E><CTRL-P>
Previous Screen	<Page Up>
Previous Tag	<CTRL-P> <F1><P>
Previous Word	<CTRL-B> <F1><B>
Query Replace	<CTRL-E><R> <CTRL-E><CTRL-R>
Redraw Screen	<CTRL-E></>
Retreat	<CTRL-R> <F1><R>
Return	<RETURN>
Right Arrow	<CTRL-K> <Right Arrow>
Save Options	<F4>
Save and Compile Buffer	<F5>
Show Current Time	<CTRL-E><T> <CTRL-E><CTRL-T>
Switch To Other Window	<CTRL-E><W> <CTRL-E><CTRL-W>
Tab	<F1><I> <Tab>
Toggle Select	<CTRL-E><S>
Toggle Syntax Checking	<CTRL-E><@>
Top	<CTRL-T> <F1><Page Up> <F1><T>

Table D-2: ANSI Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Keys(s) To Press
Top of Window	<F1><CTRL-U> <F1><Up Arrow>
Two Windows	<CTRL-E><2>
Undo	<CTRL-X> <F1><X>
Up Arrow	<CTRL-U> <Up Arrow>
Use Buffer	<CTRL-E><U> <CTRL-E><CTRL-U>
View Intermediate Source	<CTRL-E><V> <CTRL-E><CTRL-V>
Window Options	<F3>
Word Capitalize	<CTRL-E><->
Word Lowercase	<CTRL-E><_>
Word Uppercase	<CTRL-E><^>

CIT-500

Table D-3: CIT-500 Keyboard Mapping for the Full Screen Editor

Editor Action	Keys(s) To Press
Help Menu	<F15>
Advance	<CTRL-A> <PF1><A>
Begin Select	<Home>
Beginning of Area	<PF1><CTRL-H> <PF1><Left Arrow>
Bottom	<PF1><B> <PF1><Center>
Bottom of Window	<PF1><CTRL-J> <PF1><Down Arrow>

Table D-3: CIT-500 Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Keys(s) To Press
Break	<CTRL-C>
Buffer Options	<CTRL-E><B> <CTRL-E><CTRL-B>
Copy	<CTRL-E><D> <CTRL-E><CTRL-D>
Cut	<Underscore>
Cut or Paste	<CTRL-E><C> <CTRL-E><CTRL-C>
Delete Back Character	<Delete>
Delete Character	<CTRL-D>
Delete Word	<CTRL-W> <PF1><W>
Do Editor Action	<CTRL-^> <CTRL-E><CTRL-A> <CTRL-E><a> <PF1><^>
Down Arrow	<CTRL-J> <Down Arrow>
End Select	<PF1><Home>
End of Area	<PF1><CTRL-K> <PF1><Right Arrow>
Enhance	<CTRL-G> <PF1>
Erase Area	<CTRL-L>
Erase to Beginning	<PF1><Delete>
Find	<CTRL-E><F> <CTRL-E><CTRL-F>
General Help	<CTRL-Z> <PF1><Y> <PF1><Z> <CTRL-E><?> <CTRL-E><F15>
GETOUTALL	<PF1><F20> <PF1><PF4>
Goto Tag	<CTRL-E><G> <CTRL-E><CTRL-G>
Enlarge Current Window	<CTRL-E><+>

Table D-3: CIT-500 Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Keys(s) To Press
Field Help	<PF2>
Last Buffer	<CTRL-E><L> <CTRL-E><CTRL-L>
Left Arrow	<CTRL-H> <Left Arrow>
List Buffers	<CTRL-E><F9>
Macro Definition	<PF1><K>
Make Buffer	<PF1><F16> <CTRL-E><=> <CTRL-E><[>
Mark Options	<CTRL-E><M> <CTRL-E><CTRL-M>
Next Find	<F9> <PF1><PF2> <CTRL-E><N> <CTRL-E><CTRL-N>
Next Screen	<Center> <PF1><N>
Next Tag	<CTRL-N> <PF1><N>
Next Word	<CTRL-F>
One Window	<CTRL-E><1>
Only Save Buffer	<F19>
Other Options	<CTRL-E><O>
Paste	<EOL> <PF1><EOL>
Previous Find	<CTRL-E><P> <CTRL-E><CTRL-P>
Previous Screen	<EOP> <PF1><P>
Previous Tag	<PF1><P>
Previous Word	<CTRL-B>
Query Replace	<CTRL-E><R> <CTRL-E><CTRL-R>
Redraw Screen	<CTRL-E></>

Table D-3: CIT-500 Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Keys(s) To Press
Retreat	<CTRL-R> <PF1><R>
Return	<RETURN>
Right Arrow	<CTRL-K> <Right Arrow>
Save Options	<F20> <PF4>
Save and Compile Buffer	<F16> <CTRL-E><F16>
Show Current Time	<CTRL-E><T> <CTRL-E><CTRL-T>
Switch To Other Window	<CTRL-E><W> <CTRL-E><CTRL-W>
Tab	<PF1><I> <PF1><Tab> <Tab>
Toggle Select	<CTRL-E><S>
Top	<PF1><EOP> <PF1><T>
Top of Window	<PF1><CTRL-U> <PF1><Up Arrow>
Two Windows	<CTRL-E><2>
Undo	<CTRL-X> <PF1><Underscore> <PF1><X>
Up Arrow	<CTRL-U> <Up Arrow>
Use Buffer	<CTRL-E><U> <CTRL-E><CTRL-U>
Window Options	<PF3>
Word Capitalize	<CTRL-E><->
Word Lowercase	<CTRL-E><_>
Word Uppercase	<CTRL-E><^>

Dasher

Table D-4: Dasher Keyboard Mapping for the Full Screen Editor

Editor Action	Keys(s) To Press
Help Menu	<F10>
Beginning of Area	<C1><Left Arrow>
Bottom	<C1><B> <C1><F12>
Bottom of Window	<C1><Down Arrow>
Break	<CTRL-C>
Buffer Options	<CTRL-E><CTRL-B> <CTRL-E><B>
Copy	<CTRL-E><CTRL-D> <CTRL-E><D>
Cut or Paste	<CTRL-E><CTRL-C> <CTRL-E><C>
Delete Back Character	<Delete>
Delete Character	<CTRL-D>
Delete Word	<CTRL-K>
Do Editor Action	<CTRL-E><CTRL-A> <CTRL-E><A>
Down Arrow	<Down Arrow>
End of Area	<C1><Right Arrow>
End Select	<CTRL-E><B>
Enhance	<C1> <CTRL-G>
Enlarge Current Window	<CTRL-E><+>
Erase Area	<CTRL-L> <Erase Page>
Erase to Beginning	<C1><Delete>
Field Help	<C2>
Find	<F11> <CTRL-E><F> <CTRL-E><CTRL-F>
General Help	<CTRL-E><?>

Table D-4: Dasher Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Keys(s) To Press
GETOUTALL	<C1><C4>
Goto Tag	<CTRL-E><CTRL-G> <CTRL-E><G>
Last Buffer	<CTRL-E><CTRL-L> <CTRL-E><L>
Left Arrow	<Left Arrow>
Make Buffer	<CTRL-E><=> <CTRL-E><[>
Mark Options	<CTRL-E><CTRL-M> <CTRL-E><M>
Next Find	<CTRL-E><CTRL-N> <CTRL-E><N>
Next Screen	<F12>
Next Word	<CTRL-F>
One Window	<CTRL-E><1>
Other Options	<CTRL-E><O>
Previous	<C4>
Previous Find	<CTRL-E><CTRL-P> <CTRL-E><P>
Previous Screen	<F13>
Previous Tag	<CTRL-P>
Previous Word	<CTRL-B>
Query Replace	<CTRL-E><R> <CTRL-E><CTRL-R>
Redraw Screen	<CTRL-E></>
Return	<New Line>
Right Arrow	<Right Arrow>
Show Current Time	<CTRL-E><CTRL-T> <CTRL-T>
Switch to Other Window	<CTRL-E><CTRL-W> <CTRL-E><W>
Tab	<C1><I> <C1><Tab> <Tab>
Toggle Select	<CTRL-E><S>

Table D-4: Dasher Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Keys(s) To Press
Top	<C1><F13> <C1><T>
Top of Window	<C1><Up Arrow>
Two Windows	<CTRL-E><2>
Undo	<C1><X>
Up Arrow	<Up Arrow>
Use Buffer	<CTRL-E><CTRL-U> <CTRL-E><U>
Window Options	<C3>
Word Capitalize	<CTRL-E><->
Word Lowercase	<CTRL-E><_>
Word Uppercase	<CTRL-E><^>

IBM 3151 ASCII Display Station

Table D-5: IBM 3151 Key Mapping for the Full Screen Editor

Editor Action	Key(s) To Press
Advance	<F1><A> <CTRL-A>
Begin Select	<F8>
Beginning of Area	<F1><Left Arrow>
Bottom	<F1><B> <CTRL-V>
Bottom of Window	<F1><Down Arrow> <F1><CTRL-J>
Break	<CTRL-C>
Buffer Options	<CTRL-E><B> <CTRL-E><CTRL-B>

Table D-5: IBM 3151 Key Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Copy	<CTRL-E><D> <CTRL-E><CTRL-D>
Cut	<F7>
Cut or Paste	<CTRL-E><C> <CTRL-E><CTRL-C>
Delete Back Character	<Backspace>
Delete Character	<CTRL-D>
Delete Word	<CTRL-W> <F1><W>
Do Editor Action	<CTRL-^> <CTRL-E><A> <CTRL-A> <F1><^>
Down Arrow	<CTRL-J> <Down Arrow>
End of Area	<F1><CTRL-K> <F1><Right Arrow>
Enhance	<CTRL-G> <F1>
Erase Area	<CTRL-L>
Erase to Beginning	<F1><Backspace>
Field Help	<F2>
Find	<CTRL-E><F> <CTRL-E><CTRL-F>
General Help	<CTRL-Z> <CTRL-E><Help> <CTRL-E><?> <F1><Y> <F1><Z>
Get Out All	<F1><F4> <F12>
Goto Tag	<CTRL-E><G> <CTRL-E><CTRL-G>
Enlarge Current Window	<CTRL-E><+>
Last Buffer	<CTRL-E><L> <CTRL-E><CTRL-L> <F1><L>

Table D-5: IBM 3151 Key Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Left Arrow	<Left Arrow>
List Buffers	<CTRL-E><Find>
Macro Definition	<F1><K>
Make Buffer	<CTRL-E><=> <CTRL-E><[> <F1><F12>
Mark Options	<CTRL-E><M> <CTRL-E><CTRL-M>
Next Find	<CTRL-E><N> <CTRL-E><CTRL-N> <F9>
Next Tag	<CTRL-N> <F1><N>
Next Word	<CTRL-F>
One Window	<CTRL-E><1>
Only Save Buffer	<F11>
Other Options	<CTRL-E><O>
Paste	<F6>
Previous Find	<CTRL-E><P> <CTRL-E><CTRL-P>
Previous Screen	<Home>
Previous Tag	<CTRL-P> <F1><P>
Previous Word	<CTRL-B>
Query Replace	<CTRL-E><R> <CTRL-E><CTRL-R>
Redraw Screen	<CTRL-E></>
Retreat	<CTRL-R> <F1><R>
Return	<RETURN>
Right Arrow	<CTRL-K> <Right Arrow>
Save and Compile Buffer	<F1><F> <F10> <CTRL-E><Do>

Table D-5: IBM 3151 Key Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Save Options	<F4> <F1><S>
Show Current Time	<CTRL-E><T> <CTRL-E><CTRL-T>
Switch To Other Window	<CTRL-E><W> <CTRL-W>
Tab	<F1><I> <Tab>
Toggle Select	<CTRL-E><S>
Top	<CTRL-T> <F1><T> <F1><Home>
Top of Window	<F1><CTRL-U> <F1><Up Arrow>
Two Windows	<CTRL-E><2>
Undo	<CTRL-X> <F1><F7> <F1><X>
Up Arrow	<CTRL-U> <Up Arrow>
Use Buffer	<CTRL-E><U> <CTRL-E><CTRL-U>
Window Options	<F3>
Word Capitalize	<CTRL-E><->
Word Lowercase	<CTRL-E><_>
Word Uppercase	<CTRL-E><^>

IBM PC

Table D-6: IBM PC Keyboard Mapping for the Full Screen Editor

Editor Action	Key(s) To Press
Advance	<F1><A> <CTRL-A>
Help Menu	<F1><F3>
Beginning of Area	<F1><Left Arrow>
Bottom	<F1><End> <F1><V> <F1><Page Down>
Bottom of Window	<F1><CTRL-J> <F1><CTRL-V> <F1><Down Arrow> <End>
Break	<CTRL-C>
Buffer Options	<CTRL-E><B> <CTRL-E><CTRL-B>
Copy	<CTRL-E><D> <CTRL-E><CTRL-D>
Cut	<F1><Del>
Cut or Paste	<CTRL-E><C> <CTRL-E><CTRL-C>
Delete Back Character	<Backspace> <CTRL-H> <Delete>
Delete Character	<CTRL-D> <F1><D> <DEL>
Delete Word	<CTRL-W> <F1><W>
Do Editor Action	<CTRL-^> <CTRL-E><A> <CTRL-E><CTRL-A>
Down Arrow	<CTRL-J> <CTRL-V> <Down Arrow>
End of Area	<F1><CTRL-K> <F1><Right Arrow>

Table D-6: IBM PC Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
End Select	<CTRL-E><E>
Enhance	<CTRL-G> <F1>
Erase Area	<CTRL-L>
Erase to Beginning	<F1><Backspace> <F1><CTRL-H> <F1><Delete>
Find	<CTRL-E><F> <CTRL-E><CTRL-F>
General Help	<CTRL-Y> <CTRL-Z> <F1><Y> <F1><Z> <CTRL-E><?>
Get Out All	<F1><F4>
Goto Tag	<CTRL-E><G> <CTRL-E><CTRL-G>
Enlarge Current Window	<CTRL-E><+>
Field Help	<F2> <F1><F2>
Last Buffer	<CTRL-E><L> <CTRL-E><CTRL-L>
Left Arrow	<Left Arrow>
Macro Definition	<F1><K>
Make Buffer	<CTRL-E><=> <CTRL-E><[>
Mark Options	<CTRL-E><M> <CTRL-E><CTRL-M>
Next Find	<CTRL-E><N> <CTRL-E><CTRL-N> <F6>
Next Screen	<PgDn>
Next Tag	<CTRL-N> <F1><N>
Next Word	<CTRL-F> <F1><F>
One Window	<CTRL-E><1>

Table D-6: IBM PC Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Other Options	<CTRL-E><O>
Paste	<Ins>
Previous Find	<CTRL-E><P> <CTRL-E><CTRL-P>
Previous Screen	<PgUp>
Previous Tag	<CTRL-P> <F1><P>
Previous Word	<F1><B> <CTRL-B>
Query Replace	<CTRL-E><R> <CTRL-E><CTRL-R>
Redraw Screen	<CTRL-E></>
Retreat	<CTRL-R> <F1><R>
Return	<Enter>
Right Arrow	<CTRL-K> <Right Arrow>
Save and Compile Buffer	<F1><F10> <F5>
Save Options	<F10> <F4>
Show Current Time	<CTRL-E><T> <CTRL-E><CTRL-T>
Switch To Other Window	<CTRL-E><W> <CTRL-E><CTRL-W>
Tab	<F1><I> <Tab>
Toggle Select	<CTRL-E><S>
Top	<CTRL-T> <F1><T> <F1><Home> <F1><PgUp>
Top of Window	<F1><Up Arrow> <F1><CTRL-U>
Two Windows	<CTRL-E><2>
Undo	<CTRL-X> <F1><X>

Table D-6: IBM PC Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Up Arrow	<Up Arrow> <CTRL-U>
Use Buffer	<CTRL-E><U> <CTRL-E><CTRL-U>
Window Options	<F3>
Word Capitalize	<CTRL-E><->
Word Lowercase	<CTRL-E><_>
Word Uppercase	<CTRL-E><^>

QUME

Table D-7: QUME Keyboard Mapping for the Full Screen Editor

Editor Action	Key(s) To Press
Help Menu	<PF1><PF3>
Advance	<CTRL-A> <PF1><A>
Beginning of Area	<PF1><CTRL-H>
Bottom of Window	<PF1><CTRL-J>
Bottom	<PF1><V> <CTRL-V>
Break	<CTRL-C>
Buffer Options	<CTRL-E><CTRL-B> <CTRL-E><B>
Copy	<CTRL-E><CTRL-D> <CTRL-E><D>
Cut or Paste	<CTRL-E><CTRL-C> <CTRL-E><C>
Delete Back Character	<CTRL-J>
Delete Word	<PF1><W> <CTRL-W>

Table D-7: QUME Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Delete Character	<PF1><D> <CTRL-D>
Do Editor Action	<CTRL-E><CTRL-A> <CTRL-E><A> <CTRL-^>
Down Arrow	<CTRL-L>
End Select	<CTRL-E><D>
End of Area	<PF1><CTRL-K> <PF1><CTRL-L>
Enhance	<CTRL-G> <PF1>
Enlarge Current Window	<CTRL-E><+>
Exchange Point and Mark	<PF4>
Find	<CTRL-E><CTRL-F> <CTRL-E><F>
General Help	<CTRL-E><?>
GETOUTALL	<PF4>
Goto Tag	<CTRL-E><CTRL-G> <CTRL-E><G>
Last Buffer	<CTRL-E><CTRL-L> <CTRL-E><L>
Left Arrow	<CTRL-H>
Make Buffer	<CTRL-E><=> <CTRL-E><[>
Mark Options	<CTRL-E><CTRL-M> <CTRL-E><M>
Next Find	<CTRL-E><CTRL-N> <CTRL-E><N>
Next Tag	<CTRL-N>
Next Word	<PF1><F> <CTRL-F>
Next Tag	<PF1><N>
One Window	<CTRL-E><1>
Other Options	<CTRL-E><O>

Table D-7: QUME Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Previous Tag	<PF1><P> <CTRL-P>
Previous Find	<CTRL-E><CTRL-P> <CTRL-E><P>
Previous Word	<PF1><B> <CTRL-B>
Query Replace	<CTRL-E><R> <CTRL-E><CTRL-R>
Redraw Screen	<CTRL-E></>
Retreat	<PF1><R> <CTRL-R>
Return	<CTRL-M>
Right Arrow	<CTRL-K>
Show Current Time	<CTRL-E><CTRL-T> <CTRL-E><T>
Switch to Other Window	<CTRL-E><CTRL-W> <CTRL-E><W>
Tab	<PF1><I> <CTRL-I>
Toggle Select	<CTRL-E><S>
Top of Window	<PF1><CTRL-U>
Top	<PF1><T> <CTRL-T>
Two Windows	<CTRL-E><2>
Undo	<PF1><X> <CTRL-X>
Up Arrow	<CTRL-U>
Use Buffer	<CTRL-E><CTRL-U> <CTRL-E><U>
Window Options	<PF3>
Word Capitalize	<CTRL-E><->
Word Lowercase	<CTRL-E><_>
Word Uppercase	<CTRL-E><^>

Sun

Table D-8: Sun Keyboard Mapping for the Full Screen Editor

Editor Action	Key(s) To Press
Advance	<CTRL-A> <F1><A>
Begin Select	<F7>
Beginning of Area	<F1><Left Arrow>
Bottom	<F1><Page Down> <F1><End>
Bottom of Window	<End> <F1><Down Arrow>
Break	<CRL-C>
Buffer Options	<CTRL-E><B> <CTRL-E><CTRL-B>
Copy	<CTRL-E><D> <CTRL-E><CTRL-D>
Cut or Paste	<CTRL-E><C> <CTRL-E><CTRL-C>
Delete Back Character	<Delete> <Backspace>
Delete Character	<Del> <CTRL-D>
Delete Word	<CTRL-W>
Do Editor Action	<CTRL-\> <CTRL-E><A> <CTRL-E><\>
Down Arrow	<Down Arrow> <CTRL-J>
End of Area	<F1><Right Arrow> <F1><CTRL-K>
End Select	<F1><F7>
Enhance	<F1> <CTRL-G>
Erase Area	<CTRL-L> <F1><Del>
Erase to Beginning	<F1><Delete>

Table D-8: Sun Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Find	<CTRL-E><F> <CTRL-E><CTRL-F>
General Help	<CTRL-Y> <CTRL-E><?>
GETOUTALL	<F10> <F1><F10> <F1><F4>
Goto Tag	<CTRL-E><G> <CTRL-E><CTRL-G>
Help Menu	<Help> <F1><F3>
Enlarge Current Window	<CTRL-E><+>
Field Help	<F2>
Last Buffer	<CTRL-E><L> <CTRL-E><CTRL-L>
Left Arrow	<Left Arrow> <CTRL-H>
List Buffers	<CRTL-E><F6>
Macro Definition	<F1><K>
Make Buffer	<CTRL-E><=> <CTRL-E><[>
Mark Options	<CTRL-E><M> <CTRL-E><CTRL-M>
Next Error in Buffer	<CTRL-E><E> <CTRL-E><CTRL-E>
Next Find	<F6> <CTRL-E><N> <CTRL-E><CTRL-N>
Next Screen	<Page Down> <F1><N>
Next Word	<CTRL-F>
Next Tag	<CTRL-N>
One Window	<CTRL-E><1>
Other Options	<CTRL-E><O>
Only Save Buffer	<F9>

Table D-8: Sun Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Paste	<Insert> <F1><Insert>
Previous Find	<CTRL-E><P> <CTRL-E><CTRL-P>
Previous Screen	<Page Up> <F1><P>
Previous Word	<CTRL-B> <F1><B>
Previous Tag	<CTRL-P>
Query Replace	<CTRL-E><R> <CTRL-E><CTRL-R>
Redraw Screen	<CTRL-E></>
RETURN	<RETURN>
Retreat	<CTRL-R>
Right Arrow	<Right Arrow> <CTRL><K>
Save Options	<F4>
Save and Compile Buffer	<F5> <F1><F>
Show Current Time	<CTRL-E><T> <CTRL-E><CTRL-T>
Switch To Other Window	<CTRL-E><W> <CTRL-E><CTRL-W>
TAB	<Tab> <F1><I>
Toggle Select	<CTRL-E><S> <CTRL-E><CTRL-S>
Toggle Syntax Checking	<CTRL-E><@>
Top	<F1><Page Up> <F1><Home> <CTRL><T>
Top of Window	<F1><Up Arrow> <Home> <F1><CTRL-U>
Two Windows	<CTRL-E><2>
Up Arrow	<Up Arrow> <CTRL-U>

Table D-8: Sun Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Undo	<CTRL-X>
View Intermediate Source	<CTRL-E><V> <CTRL-E><CTRL-V>
Use Buffer	<CTRL-E><U> <CTRL-E><CTRL-U>
Window Options	<F3>
Word Capitalize	<CTRL-E><->
Word Lowercase	<CTRL-E><_>
Word Uppercase	<CTRL-E><^>

Televideo 905

Table D-9: Televideo 905 Key Mapping for Full Screen Editor

Editor Action	Key(s) To Press
Advance	<CTRL-A> <F1><A>
Beginning of Area	<F1><Left Arrow>
Bottom	<F1><V>
Bottom of Window	<F1><Down Arrow>
Break	<CTRL-C>
Buffer Options	<CTRL-E><B> <CTRL-E><CTRL-B>
Copy	<CTRL-E><D> <CTRL-E><CTRL-D>
Cut or Paste	<CTRL-E><C> <CTRL-E><CTRL-C>
Delete Back Character	<Del>
Delete Character	<CTRL-D> <F1><D>

Table D-9: Televideo 905 Key Mapping for Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Delete Word	<CTRL-W> <F1><W>
Do Editor Action	<CTRL><^> <CTRL-E><a> <CTRL-E><CTRL-A>
Down Arrow	<Down Arrow> <CTRL-v>
End of Area	<F1><Right Arrow>
Enhance	<F1> <CTRL-G>
Erase Area	<Line Erase>
Erase to Beginning	<F1><Del>
Erase to End	<CTRL-U>
Find	<CTRL-E><F> <CTRL-E><CTRL-F>
General Help	<F2>
GETOUTALL	<F1><F4>
Goto Tag	<CTRL-E><G> <CTRL-E><CTRL-G>
Help Menu	<F1><F3>
Enlarge Current Window	<CTRL-E><+>
Field Help	<PF2>
Left Arrow	<Left Arrow>
Make Buffer	<CTRL-E><=> <CTRL-E><[>
Next Screen	<F1><N>
Next Word	<CTRL-F> <F1><F>
Next Tag	<CTRL-N>
One Window	<CTRL-E><1>
Other Options	<CTRL-E><O>
Paste	<Line Insert>
PREVIOUS	<F4>
Previous Screen	<F1><P>

Table D-9: Televideo 905 Key Mapping for Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Previous Word	<CTRL-B> <F1><B>
Previous Tag	<CTRL-F>
Query Replace	<CTRL-E><R> <CTRL-E><CTRL-R>
Redraw Screen	<CTRL-E></>
RETURN	<RETURN>
Retreat	<CTRL-R> <F1><R>
Right Arrow	<Right Arrow>
Save Options	<F1><F>
Save and Compile Buffer	<Esc><Esc>
TAB	<Tab> <F1><I>
Toggle Select	<CTRL-E><S>
Top	<F1><T> <CTRL-T>
Top of Window	<F1><Up Arrow>
Two Windows	<CTRL-E><2>
Up Arrow	<Up Arrow> <CTRL-K>
Undo	<CTRL-X> <F1><X>
Window Options	<F3>
Word Capitalize	<CTRL-E><->
Word Lowercase	<CTRL-E><_>
Word Uppercase	<CTRL-E><^>

DEC VT 100

Table D-10: VT100 Key Mapping for the Full Screen Editor

Editor Action	Key(s) To Press
Help Menu	<PF1><PF3>
Advance	<CTRL-A> <PF1>
Beginning of Area	<PF1><Left Arrow> <PF1><CTRL-H>
Bottom	<PF1><B>
Bottom of Window	<PF1><Down Arrow> <PF1><CTRL-J>
Break	<CTRL-C>
Buffer Options	<CTRL-E><B> <CTRL-E><CTRL-B>
Copy	<CTRL-E><CTRL-D> <CTRL-E><D>
Cut or Paste	<CTRL-E><CTRL-C> <CTRL-E><C>
Delete Back Character	<DELETE>
Delete Character	<CTRL-D>
Delete Word	<CTRL-W>
Do Editor Action	<CTRL-^> <CTRL-E><CTRL-A> <CTRL-E><A>
Down Arrow	<CTRL-J> <Down Arrow>
End of Area	<PF1><Right Arrow> PF1><CTRL-K>
End Select	<CTRL-E><E>
Enhance	<CTRL-G> <PF1>
Enlarge Current Window	<CTRL-E><+>
Erase Area	<CTRL-L>
Erase to Beginning	<PF1><Del>
Field Help	<PF2>

Table D-10: VT100 Key Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Find	<CTRL-E><CTRL-F> <CTRL-E><F>
General Help	<CTRL-Z> <PF1><K> <PF1><Y> <PF1><Z> <CTRL-E><?>
Get Out All	<PF1><PF4>
Goto Tag	<CTRL-E><CTRL-G> <CTRL-E><G>
Last Buffer	<CTRL-E><CTRL-L> <CTRL-E><L>
Left Arrow	<CRTL-H> <Left Arrow>
Make Buffer	<PF1><PF2> <CTRL-E><=> <CTRL-E><[>
Mark Options	<CTRL-E><CTRL-M> <CTRL-E><M>
Next Find	<CTRL-E><CTRL-N> <CTRL-E><N>
Next Tag	<PF1><N> <CTRL-N>
Next Word	<CTRL-F>
One Window	<CTRL-E><1>
Other Options	<CTRL-E><O>
Previous Find	<CTRL-E><CTRL-P> <CTRL-E><P>
Previous Tag	<PF1><P>
Previous Word	<CTRL-B>
Query Replace	<CTRL-E><R> <CTRL-E><CTRL-R>
Redraw Screen	<CTRL-E></>
Retreat	<CTRL-R> <PF1><R>
Return	<RETURN>

Table D-10: VT100 Key Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Right Arrow	<CTRL-K> <Right Arrow>
Save Options	<PF1><S> <PF4>
Show Current Time	<CTRL-E><CTRL-T> <CTRL-E><T>
Switch to Other Window	<CTRL-E><CTRL-W> <CTRL-E><W>
Tab	<PF1><I> <PF1><Tab> <Tab>
Toggle Select	<CTRL-E><S>
Top	<PF1><T>
Top of Window	<PF1><Up Arrow> <PF1><U>
Two Windows	<CTRL-E><2>
Undo	<PF1><X>
Up Arrow	<CTRL-U> <Up Arrow>
Use Buffer	<CTRL-E><CTRL-U> <CTRL-E><U>
Window Options	<PF3>
Word Capitalize	<CTRL-E><->
Word Lowercase	<CTRL-E><_>
Word Uppercase	<CTRL-E><^>

DEC VT 200

Table D-11: VT200 Keyboard Mapping for the Full Screen Editor

Editor Action	Key(s) To Press
Advance	<CTRL-A> <PF1><A>
Help Menu	<Help>
Begin Select	<Select>
Beginning of Area	<PF1><CTRL-H> <PF1><Left Arrow>
Bottom	<PF1><B> <PF1><Next Screen>
Bottom of Window	<PF1><CTRL-J> <PF1><Down Arrow>
Break	<CTRL-C>
Buffer Options	<CTRL-E><B> <CTRL-E><CTRL-B>
Copy	<CTRL-E><D> <CTRL-E><CTRL-D>
Cut	<Remove>
Cut or Paste	<CTRL-E><C> <CTRL-E><CTRL-C>
Delete Back Character	<Delete>
Delete Character	<CTRL-D>
Delete Word	<CTRL-W> <PF1><W>
Do Editor Action	<CTRL-\> <CTRL-^> <PF1><^> <CTRL-E><CTRL-A> <CTRL-E><A>
Down Arrow	<CTRL-J> <Down Arrow>
End Select	<PF1><Select>
End of Area	<PF1><CTRL-K> <PF1><Right Arrow>

Table D-11: VT200 Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Enhance	<CTRL-G> <PF1>
Erase Area	<CTRL-L>
Erase to Beginning	<PF1><Delete>
Find	<CTRL-E><F> <CTRL-E><CTRL-F>
General Help	<CTRL-Z> <PF1><Y> <PF1><Z> <CTRL-E><?> <CTRL-E><Help>
Get Out All	<PF1><F20> <PF1><PF4>
Goto Tag	<CTRL-E><G> <CTRL-E><CTRL-G>
Enlarge Current Window	<CTRL-E><+>
Help	<PF2>
Last Buffer	<CTRL-E><L> <CTRL-E><CTRL-L>
Left Arrow	<CTRL-H> <Left Arrow>
List Buffers	<CTRL-E><Find>
Macro Definition	<PF1><K>
Make Buffer	<PF1><Do> <CTRL-E><=> <CTRL-E><[>
Mark Options	<CTRL-E><M> <CTRL-E><CTRL-M>
Next Find	<Find> <PF1><PF2> <CTRL-E><N> <CTRL-E><CTRL-N>
Next Screen	<Next Screen> <PF1><N>
Next Tag	<CTRL-N> <PF1><N>
Next Word	<CTRL-F>

Table D-11: VT200 Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
One Window	<CTRL-E><1>
Only Save Buffer	<F19>
Other Options	<CTRL-E><O>
Paste	<Insert Here> <PF1><Insert Here>
Previous Find	<CTRL-E><P> <CTRL-E><CTRL-P>
Previous Screen	<PF1><P> <Prev Screen>
Previous Tag	<PF1><P>
Previous Word	<CTRL-B>
Query Replace	<CTRL-E><R> <CTRL-E><CTRL-R>
Redraw Screen	<CTRL-E></>
Retreat	<CTRL-R> <PF1><R>
Return	<RETURN>
Right Arrow	<CTRL-K> <Right Arrow>
Save Options	<F20> <PF4>
Save and Compile Buffer	<Do> <CTRL-E><Do>
Show Current Time	<CTRL-E><T> <CTRL-E><CTRL-T>
Switch To Other Window	<CTRL-E><W> <CTRL-E><CTRL-W>
Tab	<PF1><I> <PF1><Tab> <Tab>
Toggle Select	<CTRL-E><S>
Top	<PF1><Prev Screen> <PF1><T>
Top of Window	<PF1><CTRL-U> <PF1><Up Arrow>
Two Windows	<CTRL-E><2>

Table D-11: VT200 Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Undo	<CTRL-X> <PF1><Remove> <PF1><X>
Up Arrow	<CTRL-U> <Up Arrow>
Use Buffer	<CTRL-E><U> <CTRL-E><CTRL-U>
Window Options	<PF3>
Word Capitalize	<CTRL-E><->
Word Lowercase	<CTRL-E><_>
Word Uppercase	<CTRL-E><^>

DEC VT 220

Table D-12: VT220 Keyboard Mapping for the Full Screen Editor

Editor Action	Key(s) To Press
Advance	<PF1><A> <CTRL-A>
Help Menu	<Help>
Begin Select	<Select>
Beginning of Area	<PF1><CTRL-H> <PF1><Left Arrow>
Bottom	<PF1><B> <PF1><Next Screen>
Bottom of Window	<PF1><CTRL-J> <PF1><Down Arrow>
Break	<CTRL-C>
Buffer Options	<CTRL-E><B> <CTRL-E><CTRL-B>

Table D-12: VT220 Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Copy	<CTRL-E><D> <CTRL-E><CTRL-D>
Cut	<Remove>
Cut or Paste	<CTRL-E><C> <CTRL-E><CTRL-C>
Delete Back Character	<DEL>
Delete Character	<CTRL-D>
Delete Word	<CTRL-W> <PF1><W>
Do Editor Action	<CTRL-^> <PF1><^> <CTRL-E><A> <CTRL-E><CTRL-A>
Down Arrow	<CTRL-J> <Down Arrow>
End of Area	<PF1><CTRL-K> <PF1><Right Arrow>
End Select	<PF1><Select>
Enhance	<CTRL-G> <PF1>
Erase Area	<CTRL-L>
Erase to Beginning	<PF1><Delete>
Field Help	<PF2>
Find	<CTRL-E><F> <CTRL-E><CTRL-F>
General Help	<CTRL-Z> <CTRL-E><Help> <CTRL-E><?> <PF1><Y> <PF1><Z>
GETOUTALL	<PF1><PF4>
Goto Tag	<CTRL-E><G> <CTRL-E><CTRL-G>
Enlarge Current Window	<CTRL-E><+>
Last Buffer	<CTRL-E><L> <CTRL-E><CTRL-L> <PF1><L>

Table D-12: VT220 Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Left Arrow	<Left Arrow> <CTRL-H>
List Buffers	<CTRL-E><Find>
Macro Definition	<PF1><K>
Make Buffer	<PF1><Do> <CTRL-E><[> <CTRL-E><=>
Mark Options	<CTRL-E><M> <CTRL-E><CTRL-M>
Next Find	<CTRL-E><N> <CTRL-E><CTRL-N> <Find> <PF1><PF2>
Next Tag	<CTRL-N> <PF1><N>
Next Screen	<PF1><N> <Next Screen>
Next Word	<CTRL-F>
One Window	<CTRL-E><1>
Only Save Buffer	<F19>
Other Options	<CTRL-E><O>
Paste	<Insert Here> <PF1><Insert Here>
Previous Find	<CTRL-E><P> <CTRL-E><CTRL-P>
Previous Screen	<PF1><P> <Prev Screen>
Previous Tag	<PF1><P>
Previous Word	<CTRL-B>
Query Replace	<CTRL-E><R> <CTRL-E><CTRL-R>
Redraw Screen	<CTRL-E></>
Retreat	<CTRL-R> <PF1><R>
Return	<RETURN>

Table D-12: VT220 Keyboard Mapping for the Full Screen Editor (Continued)

Editor Action	Key(s) To Press
Right Arrow	<CTRL-K> <Right Arrow>
Save and Compile Buffer	<Do> <CTRL-E><Do> <PF1><F>
Save Options	<PF4> <PF1><S>
Show Current Time	<CTRL-E><T> <CTRL-E><CTRL-T>
Switch To Other Window	<CTRL-E><W> <CTRL-E><CTRL-W>
Tab	<Tab> <PF1><Tab> <PF1><I>
Toggle Select	<CTRL-E><S>
Top	<PF1><T> <PF1><Prev Screen>
Top of Window	<PF1><CTRL-U> <PF1><Up Arrow>
Two Windows	<CTRL-E><2>
Undo	<CTRL-X> <PF1><Remove> <PF1><X>
Up Arrow	<Up Arrow> <CTRL-U>
Use Buffer	<CTRL-E><U> <CTRL-E><CTRL-U>
Window Options	<PF3>
Word Capitalize	<CTRL-E><->
Word Lowercase	<CTRL-E><_>
Word Uppercase	<CTRL-E><^>

WYSE-60 (Native Mode)

Table D-13: WYSE-60 Key Mapping for the Full Screen Editor

Editor Action	Primary Key	Alternate Key(s)
Advance	<Enhance><A>	<CTRL-A>
Beginning of Area	<Enhance><Left Arrow>	<Enhance><CTRL-H> <Backspace>
Bottom of List	<Enhance><Page Down>	<Enhance>-V <End>
Bottom of Window	<Enhance><Down Arrow>	<Enhance><CTRL-J>
Break	<CTRL-C>	
Buffer Options	<CTRL-E>/B	<CTRL-E><CTRL-B>
Copy	<CTRL-E>/D	<CTRL-E><CTRL-D>
Cut	<Delete>	
Cut or Paste	<CTRL-E>/C	<CTRL-E><CTRL-C>
Delete Word	<CTRL-W>	
Delete Previous Character	<Del>	
Delete Character	<CTRL-D>	
Do Editor Action	<Ins>	<Enhance>-^
Down Arrow	<Down Arrow>	<CTRL-J>
End of Area	<Enhance><Right Arrow>	
End Select	<CTRL-E>/E	
Enhance	<PF1>	<CTRL-G>
Erase Area	<PF12>	<CTRL-U>
Erase to Beginning	<Enhance><Del>	
Explain	<PF2>	
Find	<CTRL-E>/F	<CTRL-E><CTRL-F>
GETOUT	<PF10>	<Enhance><F>
GETOUTALL	<Enhance><PF10>	<Enhance><PF4>
General Help	<Enhance><K>	<CTRL-Z>
Goto Tag	<CTRL-E>/G	<CTRL-E><CTRL-G>
Grow Current Window	<CTRL-E>/+	

Table D-13: WYSE-60 Key Mapping for the Full Screen Editor (Continued)

Editor Action	Primary Key	Alternate Key(s)
Help Menu	<Enhance><PF3>	
Insert/Typeover Toggle	<Ins>	<Enhance><Ins>
Key Help	<Enhance><PF2>	
Last Buffer	<CTRL-E>/L	<CTRL-E><CTRL-L>
Left Arrow	<Left Arrow>	<CTRL-H> <Backspace>
Make Buffer	<CTRL-E> =	<CTRL-E> [
Mark Options	<CTRL-E> M	<CTRL-E><CTRL-M>
Next Find	<PF6>	<CTRL-E> <N>
Next Screen	<Page Down>	
Next Tag	<CTRL-N>	<Enhance><N>
Next Word	<CTRL-F>	
Other Options	<CTRL-E> O	
One Window	<CTRL-E> 1	
Paste	<Insert>	<Enhance><Insert> <Enhance><G>
Previous Find	<CTRL-E> P	<CTRL-E><CTRL-P>
Previous Screen	<Page Up>	
Previous Tag	<Enhance><P>	
Previous Word	<CTRL-B>	
Query Replace	<CTRL-E> R	<CTRL-E><CTRL-R>
Redraw Screen	<CTRL-E> /	
Retreat	<CTRL-R>	<Enhance><R>
RETURN	<Return>	
Right Arrow	<Right Arrow>	<CTRL-L>
SAVE Options	<PF4>	<PF9 <>Enhance><S>
Save and Compile Buffer	<PF5>	<Esc><Esc>
Show Current Time	<CTRL-E> <T>	<CTRL-E><CTRL-T>
Switch to Other Window	<CTRL-E> <W>	<CTRL-E> <CTRL-W>
Tab	<Tab>	
Toggle Select	<CTRL-E> <Y>	

Table D-13: WYSE-60 Key Mapping for the Full Screen Editor (Continued)

Editor Action	Primary Key	Alternate Key(s)
Top of Window	<Enhance><Up Arrow>	<Enhance>-<CTRL-K>
Top	<Enhance><Page Up>	<Enhance><T>
Two Windows	<CTRL-E> <2>	
Undo	<CTRL-X>	<Enhance>-<CTRL-X>
Up Arrow	<Up Arrow>	<CTRL-K>
Use Buffer	<CTRL-E> U	<CTRL-E><CTRL-U>
Window Options	<PF3>	
Word Capitalize	<CTRL-E> -	
Word Lowercase	<CTRL-E> _	
Word Uppercase	<CTRL-E> ^	

Glossary of Terms

Application

A set of forms, reports, and other objects and programs linked together by menus to form a structure that provides a working user interface to a database.

Application Help Facility

A suite of help features and utilities that allows you to develop and deploy a context-sensitive on-line help system for your Open M/SQL applications.

Application Mode

The mode in which application end-users interact with a database using forms and windows designed by the application developer. In application mode, users enter the application directly from the operating system prompt and never see the M prompt. Navigation is typically guided by a menu structure designed by the application developer.

Application Program Interface (API)

Software on your **client** machine that handles the interface between an application tool or C application and the **server**, and manages communications with the server. The Relational Client supports two APIs, the ORACLE Call Interface (OCI) and Microsoft's Open Database Connectivity (ODBC).

Auxiliary Window

Any window defined for a form that is neither the form's master window nor its row selection window, i.e., any window in a form that is not the first window to appear when the form is invoked. Forms may have multiple auxiliary windows.

Base Table

A collection of data represented in a simple 2-dimensional format consisting of one or more rows with one or more columns. Each row has at most one value for each column, and each row is unique, meaning that it differs by at least one column value from every other row. Base tables are mapped directly to physical storage structures.

Branching Field

A form-only field of data type Branching whose sole purpose is to serve as a mask for an underlying trigger action, which typically branches to another window within the current form or to the master window of another form. Branching fields can also invoke other actions besides a window branch. On a window, branching fields are represented by their captions, which always appear enclosed in brackets.

Caller ID

Identifier that allows each menu item or form trigger that calls a form, window, or menu to identify itself to the called object. The Caller ID is stored in the {%caller} external reference variable.

Cascading Menus

Term used to describe the style of presentation for a pop-up menu that is called by another pop-up menu. Multiple pop-up menus called in succession cascade on the screen so as not to overlap.

Characteristic Relationship

A programmer-defined join between base tables in which rows in a “child table” are existence-dependent on rows in a “parent table” in a many-to-one manner—a single parent row can have many child rows.

Child Form

A form whose data source is a child table. Child forms may be single-row or multi-row and may access rows with the parent “known” or “not known”. If the parent is known, rows are called from the RowID in the parent table, which limits row selection to children of the parent table RowID. If the parent is not known, all child table rows are selected.

Child Table

A base table that is existence-dependent on another table (its parent) in a characteristic relationship. Rows in a child table must have a pointer to a row in the parent table.

Code Generation

The process by which the Open M/SQL compiler generates executable M routines from programmer specifications. This happens when you compile an Open M/SQL base table, form, menu object, or report.

Column

See Field.

Communications Protocol

A set of conventions that defines how data is transferred between computers on a network. More specifically, it is the software that determines how a message packet is formatted. Communication protocols are used to talk to a network interface device. More than one communication protocol can share the same physical interface device. The Open M/SQL Relational Client currently supports the following communication protocols: TCP/IP, DECnet (for VAX/VMS systems only), and Memory-to-Memory (for DTM systems only).

Compilation

The process in which programmer specifications are translated by the Open M/SQL compiler into macro source code and stored as .MAC routines. The macro preprocessor uses macro source code routines to produce intermediate code, which is then saved as executable M code (called object code).

Computed Field

A field whose value is derived from a calculation defined in M code. The M code can reference other fields in the associated base table as well as M functions and special variables.

Conversion Code

M code used by Open M/SQL to convert field data values from external input formats to internal storage formats and from internal storage formats to external display formats.

Cursor

An identifier for an SQL request. Once a client application has established a connection with the server, it creates a cursor. When the client application wants to send an SQL request to the server, it associates the request with a cursor. An application can have multiple cursors active within a connection. Only one SQL request is associated with a cursor at a given point in time. Once the application receives the server's response to a request, however, it can reuse the cursor for another request.

Cursor-Based SQL

A type of embedded SQL query that opens a cursor to process the query. When your application needs to access multiple rows of data, you must use a cursor. A cursor acts like a pointer—it focuses on accessing and processing one row at a time, then moves from that row to the next in the sequence.

Database

A collection of related data.

Data Dictionary

A component of Open M/SQL used to describe the elements of the relational database, including both its conceptual content and the mapping of its logical data definitions to physical data structures in the global database.

Database Field

A field that is defined in a Data Dictionary base table. A form has access to all database fields defined for its associated base table. You can even modify certain display characteristics of the field at the form level.

Database Form

Any form that uses a Data Dictionary base table as its data source.

Data Source

The Data Dictionary base table from which a database form retrieves its data. A form may have only one data source. Non-database forms do not have any data source. Forms cannot use views as their data source.

Designated Table

The Data Dictionary base table that is referenced (pointed to) by a Designative Reference field and accessed by Designative Display fields in a cross-table reference form.

Designative Display Field

A form-only field based on a Designated Reference field (defined in the associated base table) that retrieves data from (and optionally adds data to) a specified field in the designated table. Designative Display fields are the vehicles of cross-table referencing. You may create Designative Display fields for any fields in the designated table, and you may place them on windows in the form.

Designative Reference

A programmer-defined join between two base tables in which one field of the designating table contains the Row IDs of all rows in the referenced table. In relational database terminology, the designating table has a "foreign key" on the referenced table.

Developer

A component of Open M/SQL that allows application developers to use both program-level code and application generator technology to create relational database applications.

Device

A piece of hardware that is part of a computer system, such as a terminal, printer, disk drive, or magnetic tape drive.

Directory

A name for a location on a disk where files can be stored.

DTM

InterSystems' implementation of the M programming language designed to run on IBM-compatible PCs based on Intel 80386 and higher microprocessors.

Embedded SQL

SQL statements that are directly embedded within M routines at the macro source code level. These statements are prefixed by the M binding syntax, &sql.

Export/Import Utility

A window-based utility that allows you to port Open M/SQL "objects" (base tables, views, forms, reports, queries, menus, menu objects, help topics, and help documents) between different directories and different computers.

External Value

The value of a field in its external display format, i.e., after it has passed through internal-to-external conversion code. A field may have both an internal and an external value.

Fields

A named unit of data in a base table row, usually representing a real world entity, such as a name, social security number, or date of birth for that row. Also called "column" or "attribute".

Field Caption

The descriptive text attached to a field on a window.

Foreign Key

See Designative Reference.

Forms

A collection of one or more associated windows that either displays information or prompts the user to enter information, or both.

Form Generator

An application generator component of Open M/SQL that allows application developers to design highly sophisticated window-based forms that interact with the Relational Data Dictionary (or optionally an alternative data source) to add, update, retrieve, and delete database information.

Form-Only Field

A field that is created for a form and exists only at the form level, independent of the Data Dictionary. Form-only fields may appear on a form but cannot file data to the Data Dictionary. All fields on a non-database form are form-only.

Form-Only Form

A non-database form that does not file data to any data structure.

Full Screen Editor

An ISM (InterSystems' original implementation of M) utility that allows you to create, edit, and view macro source code routines, intermediate code routines, and include files.

Global

A disk-based data storage unit specified by the M programming language standard. Also called "global variables", these are commonly implemented using balanced-tree technology.

Global Database

A database in which all data is stored in a system of multiply-subscripted arrays called "globals". This is the underlying logical and physical data storage structure of an Open M/SQL database. Relational tables are mapped to the global database through the Open M/SQL Relational Data Dictionary.

Help Document

A user documentation manual for your application based on the help definitions you have already created. You create help documents using the Help Document Creation facility. A help document consists of chapter numbers and titles, paragraph/section headings and numbers, screen images of menus, screen images of forms, screen images of individual windows, long and short help messages for menu options and fields, help topic text, and optionally an automatically generated table of contents and index.

Help Document Creation Facility

An extension to the Application Help facility that allows you to create a printed documentation manual for your application based on the help definitions you have already created.

Help Text Entry Facility

An extension to the Application Help facility that centralizes access to the help attributes associated with forms and menu objects and provides an easy-to-use interface for creating help text definitions and deploying them throughout an application.

Help Topic

A block of programmer-defined help text which you can make available to end-users in a context-sensitive fashion at various points in an application by attaching it to base tables, forms, windows, menu objects, and menu object choices or by enabling it via triggers.

Horizontal Menu

A type of old-style menu that is tied to a window on a form and displays its list of options horizontally across the bottom of the screen just below the status line of the window to which it is tied.

Implicit Join

A programmer-defined join between related tables defined in the Data Dictionary that allows you to query multiple tables without specifying data access restrictions in the WHERE clause. Implicit joins can designate characteristic relationships or designative references.

Include File

Files containing definitions that can be used in the preprocessor phase of compilation to expand macro source routines and determine whether optional lines of code should be included. Include files can also be used to include a common block of code in several routines, saving the overhead of calls to a common subroutine.

Index Map

A map for one or more database fields that contains the Row ID of each row in a base table. Index maps speed up access to rows looked up by values for the index fields and allow rapid retrieval of rows sorted by one or more index fields. The Open M/SQL Data Dictionary automatically generates index maps for all fields specified as lookup fields in the lookup specifications for a base table (when default physical structure is used).

Integrity Constraints

Programmer-defined constraints on data insert, update, and delete operations that ensure the accuracy and completeness of the application and the underlying database.

Interactive Query Editor

Open M/SQL facility for defining and running ad hoc queries. The Interactive Query Editor provides a free-form SQL editor environment (similar to the Full Screen Editor) that enables users to define and run any syntactically valid SQL query. The Interactive Query Editor also provides full screen editing capabilities.

Intermediate Source Code

The standard 3GL M source code available in all M implementations. Intermediate code is produced from macro source code by the Open M/SQL compiler. At the intermediate code level, all preprocessor syntax, including embedded SQL, is resolved, and the routine contains only pure M source code. You can write M routines directly at the intermediate code level, but you cannot use embedded SQL or other preprocessor syntax, such as macros.

Internal Value

The value of a field as stored internally by Open M/SQL, i.e., after it has passed through external-to-internal conversion code. A field may have both an internal and an external value.

Join

A link between base tables that defines the relationship between the data in those tables.

Learn-As-You-Go (LAYGO)

The ability for a form to add new rows to a foreign base table at run time.

License

An agreement between InterSystems and its customer that defines the components of Open M software available to the customer and the number of users who can use each component. A customer must be licensed in order to run Open M. License information is distributed in a *Product Activation Key* and stored on your system in a file named MSQL.KEY.

Login

The act of signing on to a system. Database administrators and application developers log in to Open M/SQL by typing “do ^%msql” at the M prompt and then providing UserName and Password information at the Open M/SQL User Identification window.

Long Help Message

A multiple-line programmer-defined help text message associated with a field or menu object choice that displays to the screen in a run-time window when the user presses the <EXPLAIN> key twice in succession from a field or menu object choice. For fields, you may define triggers to automatically display the long help message based on certain run-time circumstances.

Lookup Display Field

Field or combination of fields whose values are displayed in the lookup box that lists all matching entries retrieved by the lookup query in row selection for single-row forms. The information displayed by these fields helps the user select the appropriate row.

Lookup Field

Field or combination of fields used by single-row forms to select rows from the database. When a form is run, the user enters lookup information into the lookup field(s), and the system runs a lookup query to match the user input against actual database values.

Lookup Specifications

The complete set of all lookup queries, including their lookup fields and lookup display fields, defined to perform row selection for a single-row form.

Lookup Query

A set of lookup fields and lookup display fields defined to perform row selection for a single-row form. A Single-row form may have multiple lookup queries.

M Language

An ANSI-Standard procedural programming language specifically designed for database applications. M is the foundation technology of InterSystems' entire Open M product line.

M Database

A MUMPS.DAT file and, on server systems that support multivolume databases, from 0 through 7 MUMPS.EXT files.

Macro Preprocessor

Phase of the Open M/SQL Compiler that converts macro source code, which may include macro constructs and embedded SQL, into intermediate code, which is pure M source code. The macro preprocessor resolves macros and code-generates embedded SQL.

Macro Source Code

The highest, most flexible and permissive level of code at which routines can be written. Macro source code permits the definition of macros and embedded SQL statements using a combination of ANSI-Standard M syntax, special macro preprocessor commands, and ANSI-Standard SQL.

Map

A map describes the relationship of the logical structure of a base table to the physical structure of the underlying global database. The Open M/SQL Relational Data Dictionary uses maps to describe an M global database in relational terms.

Master Map

Each base table has exactly one master map. The master map defines the global structure for all of the data fields in the base table. In M terminology, the master map defines the “upright file”.

Master Window

The introductory window of any form. Every form must have a master window in order to be compiled and run. No form can have more than one master window.

Menu

A list of one or more choices, each of which invokes an action that performs a specific task. Menus can display their options vertically (top to bottom) or horizontally (left to right). Menus are typically used to unite various components of an application and provide access to them from a central location.

Menu Generator

An application generator component of Open M/SQL that allows application developers to design and maintain menus for an application. Menus unite the various components of an application in a logical and visually sophisticated manner and structure an application by defining how it is organized and how it is presented to users. You can design menu objects or old-style menus.

Menu Object

A list of one or more choices, each of which invokes an action that performs a specific task. A menu object can be run as either a menu bar or a pop-up menu. Menu objects emulate the style of a graphical user interface (GUI) environment by providing pull-down menu bar and pop-up menu capabilities.

Menu Bar

One of the two run-time modes for a menu object, a menu bar displays its options horizontally across the screen. Menu bars can be attached to windows (in which case they appear at the top of the window), they can be attached to forms (in which case they appear at the top of the screen for all windows in the form), or they can be run as stand-alone objects (in which case they can be positioned anywhere on the screen).

M/PACT

The report writer component of Open M/SQL. M/PACT lets you create and run sophisticated end-user data reporting applications that interact with the Open M/SQL Relational Data Dictionary.

MSQL.KEY File

The file into which you must enter the encoded version of your Open M license in order to activate the license.

Multi-Row Form

A form that simultaneously presents multiple rows of data from its associated base table. The master window of a multi-row form displays its set of fields in repeating units, each unit corresponding to one database row. Multi-row forms must always have a Data Dictionary data source—they can never be non-database forms.

MUMPS.DAT File

The primary or only volume in an M database. It contains M globals and routines.

MUMPS.EXT File

A secondary volume in an M database. On Server systems that support multivolume databases, an M database can contain from 0 through 7 MUMPS.EXT files, in addition to one MUMPS.DAT file. A MUMPS.EXT file contains M globals and routines, but cannot be referenced directly except when the SYSMGR or MIS utility is used to add it to the database.

National Language Independence

An Open M/SQL internationalization feature that allows you to provide language translations for most application text, including help and error messages, display captions, and system-generated run-time messages. Open M/SQL supports eleven run-time languages.

Network

A collection of computers and connections that allows users and programs on one computer to communicate with users and programs on other computers in the network.

Network Configuration

1. A description of the location of data within a network, and of the relationships among various components in the network. The combined entries in the Hardware Description Table, the Directory Set Location Table, and the DSM-DDR Volume Set Translation Table form this description. Although multiple configurations may be defined and stored in M/NET, only one at a time can be active.
2. The actual network components and relationships.

Node

One computer in a network.

Non-Cursor-Based SQL

A type of embedded SQL query that consists of individual SELECT, INSERT, UPDATE, and DELETE statements. A non-cursor-based SQL query must always return a single row of data. Non-cursor-based SELECT statement queries are appropriate when you know that a single row of data matches the WHERE clause.

Non-Database Form

A form that is not associated with a base table in the Data Dictionary. Non-database forms may be form-only forms (no data source at all), or they may be alternative data source forms (interact with a data structure other than the Open M/SQL Data Dictionary).

Object

In Open M/SQL, an object is a structural entity that has identity and behavioral properties. Objects constitute both the structure of the databse and the structure of applications. Objects can interact with other objects. Open M/SQL object types include: base tables, views, forms, windows, fields, stand-alone captions, line objects, menu objects, menus (old-style), reports, queries, help topics, and help documents.

Object Code

The lowest level of code produced by the Open M/SQL compiler. This is the code that is actually interpreted and executed. You cannot write routines at the object code level.

Object Compile Driver Utility

Open M/SQL utility that allows application developers to define and store lists of Open M/SQL objects for serial compilation. A compilation list may include multiple objects or entire applications. When run, the utility batch-compiles the code for all objects included in the specified compilation configuration.

Object Integrity Checker Utility

Open M/SQL utility that checks the integrity of specified object definitions or groups of object definitions in the current directory and generates a report of all integrity errors that it finds. The Integrity Checker utility can also automatically correct some of the integrity errors it finds. This utility enables application developers to easily identify and repair integrity errors in their applications.

Old-style Menu

The traditional Open M/SQL horizontal and vertical menus. The Menu Generator supports the creation and deployment of both menu objects and old-style menus. For new applications, InterSystems recommends that you create menus using the more flexible and sophisticated menu object generation environment.

Open M

Name of InterSystems' complete line of products, of which the foundation technology is M.

Open M refers to the foundation software on top of which Open M/SQL runs.

Open M/SQL

Open M/SQL is the RDBMS (Relational Database Management System) component of Open M—it includes a development environment for creating advanced relational database applications, a management system for maintaining them, and a run-time environment for executing them.

Open M/SQL features an integrated development environment consisting of an advanced relational database management system, application generator, report generator, and procedural programming language. It combines the SQL relational query language with the M database-oriented procedural programming language.

Open M/SQL Relational Database

An M database that is organized in a relational structure by creation in or mapping to the Open M/SQL Relational Data Dictionary.

One-Way Outer Join

A programmer-defined join specified by using the symbol =* in place of = in the WHERE clause of an SQL query. This type of join designates the first table specified in the join condition as the source table and includes all rows from the source table in the output table, even if there is no match in the second table. The source table pulls relevant information out of the second table but never sacrifices its own rows for lack of a match in the second table.

Parent ID

Field automatically created by the Open M/SQL Relational Data Dictionary to specify the Row ID in a child table when you define a characteristic relationship between tables. The Parent ID acts like a designative reference from the child table to the parent table and has the same name as the parent table.

Pop-Up Menu

One of the two run-time modes for a menu object, a pop-up menu displays its options in a vertical list. Pop-up menus can be run as stand-alone objects, they can be pulled down from menu bars, they can be called from other pop-up menus (in which case they display in cascading style), or they can be called from within forms.

Primary Key

A field or combination of fields used to uniquely identify each row of a base table. In Open M/SQL, the function of the primary key is performed by the Row ID.

Privilege

The authority of a user to perform an action on an object. The owner of an object has the responsibility for granting and revoking privileges to users and groups of users on that object.

Process

An entity scheduled by the system software, which provides a context in which an image executes. A process is associated with certain hardware and software and uses an address space.

Product Activation Key

A paper key that arrives with your software distribution on which is printed an encoded version of your Open M/SQL license. You must enter this information into a file called MSQL.KEY in order to activate the license.

Programmer Mode

The mode in which all program development activity takes place. In programmer mode, you initiate programs from the M prompt, and the M prompt reappears at the conclusion of every program you run. Programmer mode encompasses the M environment and all programs that can be called from it, including the Open M/SQL development environment and run time environment. In programmer mode, you can create applications that users subsequently run in application mode.

Prompt

A system-generated signal requesting some user response.

Pull-Down Menu

Term used to describe the style of presentation for a pop-up menu that is called from a menu bar.

Query

An SQL language construct that allows you to extract and manipulate the data in a relational database. In the Open M/SQL relational environment, queries can be embedded directly within M code, or they can be written interactively using the Interactive Query Editor or the Query Generator (for SELECT-statement queries only).

Query-Based View

A view that is based on the output of a SELECT-statement query. You may create query-based views using the CREAE VIEW statement.

Query Generator

Open M/SQL facility for defining and running ad hoc queries. The Query Generator provides an easy-to-use template of SQL SELECT statement syntax equipped with fields for the appropriate SQL clauses, including SELECT, FROM, WHERE, ORDER BY, GROUP BY, and HAVING. This enables users to create SELECT queries by simply filling in the template. The Query Generator automatically generates all queries as a cursor-based SELECT statement queries, which means they can retrieve multiple data rows into the output table.

Referential Integrity

Referential integrity constraints ensure that database insert, update, and delete operations that apply to tables linked by implicit joins do not compromise the accuracy and completeness of the database.

Relation

A link between base tables. See **Characteristic Relationship** and **Designative Reference**.

Relational Database

A collection of related data that is organized according to the relational model.

Relational Environment

Environment in which you define the database in relational terms and use Open M/SQL's application generator tools to create, modify, and execute advanced relational database applications. The Open M/SQL relational environment combines two ANSI-standard languages—M and SQL.

Relational Gateway

A component of Open M/SQL which provides a function call-based interface that enables M applications to connect with external database servers. The external database servers can be M-based relational databases or non-M-based ("foreign") relational databases, such as Oracle. M applications act as clients to these external database servers and issue SQL requests to retrieve data.

Relational Model

The model for database management in which all data is organized in relational tables. The SQL language is based on the relational model, as is the Open M/SQL Relational Data Dictionary.

Relational Client

A set of drivers that resides on a client system and allows you to connect from an external application program to an Open M/SQL relational database using the Open M/SQL Relational Server.

Relational Server

A component of Open M/SQL that makes data stored in Open M/SQL relational databases available to applications developed in certain Windows-based tools such as Microsoft Excel, Microsoft Access, and Pilot LightShip, as well as applications developed in C or C++.

Report

A program defined using the M/PACT report writer that retrieves and displays data from the relational Data Dictionary.

Required Field

A field in a base table or on a form that must contain a valid non-null value before the row can be filed.

Row

A group of related field values that describes an entity in the domain of a relational table. For example, in a Customers table, a row describes a single customer. Also called a “record” in traditional data processing terminology, or a “tuple” in relational database terminology.

Row ID

In a base table defined in the Open M/SQL Relational Data Dictionary, the RowID is a field (or combination of fields) whose value uniquely identifies each row in the base table. A RowID field must always have a unique value. In relational terminology, the Row ID is the same as the primary key.

Row Selection

The process of selecting a row to be retrieved from the database for a single-row form. The programmer defines lookup queries to perform row selection, and the user runs a lookup query by providing the requisite lookup information.

SELECT List Item

An element in a SELECT statement that tells the server what data to retrieve. It can be either a column name or an expression.

Server

1. The system on which the Open M/SQL Relational Server and your Open M/SQL relational database resides.
2. An M process on the Relational Server system that communicates with the client API.

Server Master

A component of the server software that "listens" for connection attempts from clients connected via TCP, and spawns server processes to service those connections. Each server master is an M process.

Server Process

A JOBbed process on the server that services a single client connection. A server master creates a server process for this purpose.

Short Help Message

A one-line programmer-defined help text message associated with a field or menu object choice that displays at the bottom of the screen just below the status line when the user presses the <EXPLAIN> key from a field or menu object choice. For fields, you may define triggers to automatically display the short help message based on certain run-time circumstances, or you may enable automatic display of the short help message for all fields on a form.

Sign-on

The act of entering Open M/SQL, in order to use M, Developer, or an Open M/SQL application. See also login.

Single-Row Form

A form that presents data from its associated base table one row at a time. The users selects a row from the row selection window.

SQL (Structured Query Language)

Stands for Structured Query Language; SQL is the ANSI-Standard 4GL programming language designed specifically for accessing and maintaining relational databases.

Stand-Alone Caption

Descriptive text that appears on a window but is not attached directly to a field. Often used in the capacity of a window header.

Subquery

A subquery is an SQL SELECT statement query that is embedded within another SQL SELECT statement query. Open M/SQL permits the embedding of subqueries within the SELECT, FROM, and WHERE clauses of the outer query. Open M/SQL also permits the nesting of subqueries to any number of levels.

System Manager's Directory

The directory where the Open M/SQL database resides. This directory contains system globals, system routines, and %-utilities. You must create this directory at the DOS level. Your Open M system automatically places the Open M/SQL database in this directory during installation.

Transaction

A set of operations that forms a unit.

Trigger

A sequence of actions defined by the developer to execute at various points during an Open M/SQL application. In Open M/SQL, you can associate triggers with base tables and with forms. Base table triggers are database actions initiated by INSERT, UPDATE, or DELETE actions performed on a base table. These triggers help maintain integrity constraints and other data dependencies. Form triggers can execute at the form, window, and field levels of an application and provide numerous application navigation and processing functions, such as transferring control to different parts of an application.

Validation Code

M code used by Open M/SQL to validate field values by specifying validity constraints on field data values.

Variable

A symbolic name that is used to reference a data value. Variables can be local or global. Local variables reside in the local symbol table associated with a given partition. All Open M/SQL local variables begin with the percent sign (%). Global variables reside on disk. Some global variables in Open M/SQL also begin with the percent sign (%).

Variable Window Placement

The ability to display a window at different positions on the screen depending on the context in which it is called. Variable window placement allows you to define alternative placements for a window by specifying multiple sets of X and Y axis coordinates, each associated with an M condition that governs its execution.

Vertical Menu

A type of old-style menu that runs in stand-alone fashion and displays its options vertically in a list. Vertical menus typically serve as control flow maps for an application. The Open M/SQL Main Menu is an example of a vertical menu.

View

A virtual table created using the fields from a base table or set of base tables linked by implicit joins. Views are conceptual “windows” through which data from one or more base tables can be “viewed”.

Virtual Field

A field that does not correspond directly to a single stored value but instead is composed of several stored values. For example, the Row ID field for a child table is sometimes composed of two stored values—the Row ID of the parent and a subscript corresponding to a particular child row.

Virtual Table

A named table derived from one or more base tables that is not directly represented in physical storage. Views and query output are examples of virtual tables.

Window

A set of fields, text captions, and line objects displayed together on the screen as part of a form. Every form must have at least one window, its master window.

Word-Processing Field

A multi-line field of data type Text that has word-processing capabilities, including “automatic” line wrapping, enhanced field navigation, and the ability for users to search for a string.

Index

A

{%action} 11-4, 11-20

Additional validation code 11-24

%AFTERHAVING 9-3, 9-14

{%agg} 11-6

Aggregate functions (in SQL) 9-11
- %AFTERHAVING 9-14
- as query columns 9-11
- DISTINCT 9-14
- DISTINCT BY 9-15
- %FOREACH 9-12

%ALPHAUP 9-3, 9-41

ALPHAUP 9-38

%ALTER privilege 9-3, 9-49

ALTER VIEW statement 9-57, 10-17

Application Help facility G-1

Application mode G-1

Application Programming Interface (API) G-1

Applications G-1
- automatic generation of 1-4
- contents of 11-43
- defining the data structure 1-4
- designing 3-4
- developing forms and reports 1-5
- hand-coding in M 1-6
- inserting M code and SQL code 1-15, 11-13
- mapping the functional specifications 1-4
- mixing automatic generation and hand-coding 1-7
- portable acrros M systems 1-14
- programmer interface to 11-1
- programming methods 3-2
- strategies for developing 1-4
- system-generated routines 11-43
- tying together the various components 1-6

Arrays
- using the INTO clause to pass information into M arrays 9-19

Arrow syntax 2-19, 9-7

ASCII-Delimited format (for queries) 10-35
- contents of file 10-35
- queries created via the Query Editor 10-26
- queries created via the Query Generator 10-14
- selecting an output device 10-34

Authorization ID
- establishing from M code 11-39
- stored in %msql local variable 11-39

Auxiliary windows G-1

B

{%background} 11-7

Backups (for routines) 6-5, 6-12
- deleting 6-14, 7-29

generated by the Full Screen Editor 6-13
restoring backup versions 6-13
setting # of backup versions to be maintained 7-28
shuffling and renumbering of backups 6-12

Base table triggers 11-16
action types 11-17

Base tables 2-3, G-2
characteristic relationships 2-17, 9-9
checks made by Object Integrity Checker utility 12-24
compiled routines of 11-44
designative references 2-16, 9-7
filing entry points to compiled routines 11-45
First Normal Form 2-2
storage globals 11-45
triggers 11-16

%BEGTRANS 9-3, 9-45

Branching fields G-2

Buffers (in the Full Screen Editor) 4-16
creating 4-16
displaying in windows 4-18
loading routines into 4-17
selecting 4-16
setting marks in 4-19

C

Caller ID G-2

{%caller} 11-3, 11-20

Cartesian product operation 2-7

CASCADE option (for revoking privileges) 9-50

Cascading menus G-2

{%cellar_tuple} 11-6

Characteristic relationships 2-17, 9-9, G-2
child-to-parent relationship 2-20, 9-9
parent-to-child relationship 2-20, 9-10
specifying implict join syntax in queries 2-20, 9-9, 9-10

Charactersistic relationships
specifying implict join syntax in queries 2-20

%CHECKPRIV 9-3, 9-50, 10-17

Child forms G-2

Child tables 2-17, 9-9, G-3

CLOSE statement 8-4

Code generation G-3

Collation sequence 9-37
ALPHAUP function 9-38
changing the default collation sequence 9-42
EXACT function 9-37
field-level collation 9-39
in comparisons 9-40
Minus function 9-39
Plus function 9-39
Space function 9-39
UPPER function 9-38
using ORDER BY clause 9-40

Columns 2-2, G-3

Comment lines
in macro source code 5-15

COMMIT 9-45

Common directory 11-11

Communications protocol G-3

Compilation G-3
compiling Open M/SQL objects serially 12-5
of objects 11-43
of routines 6-11

Compilation Options window 12-8
options defined 12-9

Compiler 1-11, 5-2, 6-11
generating routines for applications 11-44

Computed fields G-3
inserting code 11-22

Concatenation operator (SQL) 9-32

Conditional maps 11-26

Configuration
Network G-13
Conversion code 9-26, G-3
external-to-internal 11-23
internal-to-external 11-23
referencing variables 11-23
Copy Query utility 10-36
Copying
queries 10-36
CREATE VIEW statement 9-57, 10-17
Curly brace syntax 11-2, 11-19
Cursor-based SQL 8-4, G-4
declaring a cursor 8-4
opening a cursor 8-4
passing information into M variables 8-5
retrieving information into a cursor 8-5
using the INTO clause 9-18
Cursors 8-4, G-4
declaring 8-4
opening 8-4
passing information into variables 8-5
retrieving information 8-5

D

^%dafmlog 11-12
Data Dictionary 1-3, 1-10, 2-5, 3-4, G-4
accessing data from 2-6
defining data structures 1-4
Data source G-4
%data(icol) 11-3
Database 1-10, 2-1, G-4
for non-ISM systems 2-4
M G-10
Open M/SQL G-15
relational 2-2
relational model G-17
Database forms G-4
{%date} 11-6
DECLARE statement 8-4
#define 5-7
DELETE privilege 9-49
DELETE queries 10-17
in non-cursor-based SQL 8-3
Designated table 2-16, 9-7, G-5
Designative display fields G-5
Designative references 2-16, 9-7, G-5
specifying implict join syntax in queries 2-19
Detailed Query Listing utility 10-40
sample report 10-42
Developer 1-3, G-5
Developer Utilities menu 12-1
accessing 12-2
Export/Import Options 12-3
invoking routine management utilities 7-3
invoking routine mangement utilities 12-40
invoking the Full Screen Editor 4-4, 12-4
National Language Reports 12-44
Object Compile Driver utility 12-5
Object Integrity Checker utility 12-23
Object Routine Prefix utility 12-42
Object String Search utility 12-35
options defined 12-3
Device G-5
Device selection (for queries) 10-32
ASCII-delimited output format 10-34
selecting a device 10-33
selecting a print format 10-33
Device Selection window 12-31
fields defined 12-32
selecting a device 12-32
selecting a print format 12-32
Directory G-5
DISTINCT 9-14
DISTINCT BY 9-15
Distributed Cache Protocol (DCP) 1-13
Distributed data processing 1-19
DROP VIEW statement 9-58, 10-17
DSM 1-14
DT Network 1-13
list of database protocols 1-13

DTM 1-14, G-5

Duplicate rows (in SQL queries) 9-16

E

%edit(icol) 11-3

#else, 5-11

#elseif, 5-11

Embedded SQL 1-15, 9-17, G-5
- cursor-based 1-15, 8-4, G-4
- example 8-10
- handling internal and external values 8-8
- handling multi-line fields 8-8
- macro references 8-7
- non-cursor-based 1-15, 8-2, G-13
- portability 8-9
- reserved tag and variable vames 8-9

#endif 5-11

Equijoins 2-11

Error messages (for SQL queries) A-3

%EXACT 9-3, 9-41

EXACT 9-37

Export/Import utility G-6
- accessing via Developer Utilities menu 12-3

Extensions (for routines) 6-3

%EXTERNAL 9-3, 9-29

External values G-6
- in SQL queries 9-26

External-to-internal conversion code 9-26, 11-23

F

FETCH statement 8-4
- retrieving information into a cursor 8-5

Field captions G-6

Fields G-6
- computed 11-22
- database G-4
- reading 11-36
- referencing in M code 11-14
- referencing within triggers 11-19
- required 11-24

{%filetype} 11-4, 11-20

%first 4-6

First Normal Form 2-2

FOR ALL operator 9-56

FOR SOME operator 9-56

%FOREACH 9-3, 9-12

Foreign key 2-16, 9-7, G-6

%FORM 9-3

Form call syntax (from M) 11-27
- calling forms by name/ID# 11-27
- calling forms by routine prefix 11-28
- parameters defined 11-28

Form Generator 1-3, 1-12, 3-4, G-6
- creating forms 1-5

Form triggers 11-17
- action types 11-17

Form-only fields G-6

Form-only forms G-7

Forms 1-5, 1-12, G-6
- checks made by Object Integrity Checker utility 12-25
- child forms G-2
- compiled routines of 11-46
- emulating form behavior in M programs 11-34
- erasing windows from screen 11-38
- form-only forms G-7
- invoking from M code 11-27
- non-database forms G-13
- single-row G-20
- storage globals 11-46
- triggers 11-17

FROM clause 10-8

Full Screen Editor 1-18, 3-2, 4-1, 12-4, G-7
- automatic date and time stamps 4-24
- automatic syntax checking 4-23
- Buffers Menu options described 4-11
- control key editing commands 4-20
- creating a new buffer 4-16
- creating routines 4-7
- cursor positioning keys 4-13

cutting and pasting 4-15
deleting text 4-14
displaying multiple buffers 4-18
edit field 4-8
editing multiple copies of a routine 4-16
editing operations 4-13
exiting 4-23
generating backup versions for routines 6-13
getting help 4-22
Help Menu options described 4-22
horizontal options menu 4-9
inserting text 4-14
invoking from Developer Utilities menu 4-4
invoking from M programmer prompt 4-3
keyboard actions D-1
loading a routine into the current buffer 4-17
loading existing routines 4-6
loading routines automatically 4-6
lockout mechanism 4-7
Mark Menu options described 4-12
navigating the menu system 4-10
Other Menu options described 4-12
overview 4-2
preventing overwrites 4-7
Primary Menu options described 4-10
replacing strings 4-20
routine types 4-2
Save Menu options described 4-23
saving routines 4-23
screen display 4-8
searching for text strings 4-20
selecting an existing buffer 4-16
setting a mark in the current buffer 4-19
Status Line 4-9
Windows Menu options described 4-11

G

Global database 1-9, G-7
accessing 1-16

Globals 11-9, G-7
base table definition 11-45
form definition 11-46
implicit 1-19
menu object definition 11-46
object definition globals 11-9
old-style menu definition 11-47
percent globals 11-12
query definition 11-47
referencing within inserted code 11-15
report definition 11-47

Grant option (for privileges)
granting 9-48
Revoking 9-50

GRANT statement 9-47, 10-17
WITH GRANT OPTION 9-48

GROUP BY clause 10-8

H

HAVING clause 10-9

Help Document Creation facility G-7

Help documents G-7

Help text
displaying in help text box 11-34
writing messages 11-35

Help Text Entry facility G-7

Help topics G-8

help^%msql function 11-34
parameters defined 11-35

Horizontal menus 1-12, G-8

I

#if, 5-11

#ifdef, 5-11

#ifundef, 5-11

Implicit globals 1-19

Implicit joins 2-15, 9-6, G-8
arrow syntax 2-19, 9-7
as characteristic relationships 2-17, 9-9
as designative references 2-16, 9-7
integrity constraints 2-21
syntax for specifying child-to-parent references 2-20, 9-9
syntax for specifying designative references 2-19

syntax for specifying parent-to-child references 2-20, 9-10

#include 5-13

Include files 1-11, 5-3, G-8
advantages of using 5-14
creating 5-2
editing 4-2
extensions 6-3
maintaining backup versions 6-5
naming 5-3

Index map G-8

Inner joins 2-11

{%inquiry_mode} 11-5, 11-20

INSERT privilege 9-49

INSERT queries 10-17
handling mutli-line fields 9-25
in non-cursor-based SQL 8-3
passing information into M arrays 9-21
with VALUES clause 8-3, 9-23

Inserted code 1-15, 11-13
referencing fields 11-14
referencing globals 11-15
referencing variables 11-15
summary of insert locations 11-13
syntax checking 11-15
types of code 11-14
within additional validation code 11-24
within computed fields 11-22
within conditional maps 11-26
within internal/external conversion code 11-23
within map subscripts and pieces 11-25
within NEXT subroutine 11-25
within override lookup queries 11-22
within required-maybe fields 11-24
within triggers 11-16

Integrity checking
see Object Integrity Checker utility

Integrity constraints G-8

Interactive Query Editor
see Query Editor

Intermediate code routines 1-17, 3-2
converting to macro source code 6-2
creating 3-3
editing (using Full Screen Editor) 4-2
editing (using the Routine Line Editor) 3-3
extensions 6-3
maintaining backup versions 6-5

Intermediate source code 1-11, G-9

%INTERNAL 9-3, 9-29

Internal values G-9
in SQL queries 9-26

Internal-to-external conversion code 9-26, 11-23

INTO clause 9-18
passing information into M arrays 9-19
passing information into mixed variables 9-20
using in cursor declaration 9-18
with cursor-based SQL 8-5
with INSERT queries 9-21
with non-cursor-based SQL 8-3
with SELECT queries 9-19
with UPDATE queries 9-22

%INTRANS 9-3, 9-45

%is 11-2

ISM 1-14

J

Join operation 2-11

Joins 9-5, G-9
equijoin 2-11
implicit joins 2-15, 9-6, G-8
inner joins 2-11
one-way outer joins 2-14, 9-5, G-15

K

Keyboards
key mapping for Full Screen Editor function keys D-1

Keys
Full Screen Editor function keys mapped to your terminal type D-1

Keywords (in SQL) 9-3
reserved words B-1

L

Language
report on language translations 12-44

Learn-As-You-Go (LAYGO) G-9

License G-9

LIKE predicate 9-34
error handling 9-36
ESCAPE qualifier 9-35
pattern matching characters 9-34
using host variables for search pattern 9-35

{%linenum} 11-5

List Queries utility 10-39
sample report 10-39

Login G-9

Long help messages G-10

Lookup display fields G-10

Lookup fields G-10

Lookup queries G-10
inserting code within override lookup queries 11-22

Lookup specifications G-10

M

M G-10
database G-10
vendor-independence 1-14

M code 1-10
calling forms 11-27
calling menu objects 11-32
calling old-style menus 11-32
calling queries 10-31, 11-31
calling reports 11-30
cleaning up windows 11-38
compiling an object compilation configuration 12-22
displaying help text in a help text box 11-34
embedded SQL 8-1
establishing Authorization ID 11-39
inserting into applications 11-13
inserting into triggers 11-21
protecting critical Open M/SQL variables 11-40
reading fields 11-36
writing message text 11-35

M database
for non-ISM systems 2-4

M directory
as schema 2-4

M operators
summary of 9-30

M programming language 1-2, 1-15
global references to the database 1-16
language processor in Open M/SQL 1-9
procedural programming 1-6

M/NET 1-13
list of database protocols 1-13

M/PACT 1-3, 1-12, G-12
generating reports 1-5

M/SQL Integrity Check Utility menu 12-29

Macro preprocessor 5-2, 5-4, G-11
commands 5-4, 5-7
functions 5-5, 5-16
macro references 5-5

Macro routine utilities
see routine management utilities

Macro source code 1-11, 5-2, G-11
advantages of using include files 5-14
#define statements 5-7
#else statements 5-11
#elseif statements 5-11
embedding SQL 5-16, 8-1
#endif statements 5-11
#if statements 5-11
#ifdef statements 5-11
#ifundef statements 5-11
#include statements 5-13
indicating comment lines 5-15
macro references 5-5
making code inter-vendor portable 5-17
nesting macros 5-9
#noshow statements 5-14
preprocessor commands 5-4
preprocessor functions 5-5
referencing include files 5-3

#show statements 5-14
##sql preprocessor function 5-16, 8-2
&sql preprocessor function 5-16, 8-2
summary of preprocessor commands 5-7
summary of preprocessor functions 5-16
#undef statements 5-9
##vendor preprocessor function 5-17

Macro source routines 1-17, 3-2, 5-1, 6-2
compiling 5-2
creating 5-2
editing 4-2
extensions 6-3
in vendor-independent environment 1-14
macro preprocessor 5-4
maintaining backup versions 6-5
naming 5-3
portability across directories 8-9
portability across M systems 5-3

Macros 5-5
nested expansion 5-9
referencing in embedded SQL 8-7

Maps G-11
inserting code for conditional 11-26
inserting code for NEXT subroutine 11-25
inserting code for subscripts and pieces 11-25

Master map G-11

Master window G-11

^mcompd 11-9

^mconv 11-9

^mdd 11-9, 11-45

^mddc 11-9, 11-45

%MENU 9-3

Menu bars 1-12, G-12

Menu call syntax (for old-style menus) 11-32
parameters defined 11-32

Menu Generator 1-6, 1-12, G-11

Menu object call syntax 11-32
calling menu objects by routine prefix 11-33
calling menu objects with menu call entry point 11-33

Menu objects 1-12, G-12
checks made by Object Integrity Checker utility 12-26
compiled routines of 11-46
horizontal menus 1-12
invoking from M code 11-32
menu bars 1-12
pop-up menus 1-12
storage globals 11-46
vertical menus 1-12

{%menubar} 11-6

{%menuid} 11-6

Menus G-11
cascading G-2

Menus (old-style) 1-12, G-14
invoking from M code 11-32
storage globals 11-47

{%menutype} 11-6

%MEUNOBJ 9-3

^mexpnew 11-9

^mexport 11-9

^mform 11-9, 11-46

^mformc 11-9, 11-46

^mhelp 11-9

Minus collation sequence function 9-39

^mlock 11-9

^mmenu 11-9, 11-47

^mmisc 11-9

^%mmsg 11-12

%mobject 11-10

^%mobject 11-12

%mode 11-2

^mpriv 11-10

^mql 11-10, 11-47

^mreport 11-10, 11-46, 11-47

^mreportc 11-10, 11-47

^mroutine 11-9

%msg 11-3

MSM 1-14

%msql 11-2
entry point for calling forms 11-27
entry point for calling old-style menus 11-32
entry point for calling queries 11-31
entry point for calling reports 11-30
entry point for cleaning up windows 11-38
entry point for displaying help text in a help text box 11-34
entry point for establishing Authorization ID 11-39
entry point for reading fields 11-36
entry point for writing message text 11-35
list of entry points 11-8

^%msql 11-12

MSQL.KEY file G-12

%msqlutl utility 11-40
displaying list of critical Open M/SQL variables 11-42
pushing Open M/SQL variables onto a stack 11-40
reinstating variables from stack 11-41

$$msqlvars^%msqlutl function 11-42

^mtemp 7-30, 11-10

^mterm 11-10

Multi-line fields 2-21
in INSERT queries 9-25
in SELECT queries 9-24
in UPDATE queries 9-25
retrieving values in SQL 8-8, 9-24

Multi-row forms G-12

MUMPS.DAT file G-12

MUMPS.EXT file G-12

^%muser 11-12

^mutil 11-10

^mxdd 11-10

^mxdoc 11-10

^mxform 11-10

^mxhtop 11-10

^mxmenob 11-10

^mxmenu 11-10

^mxql 11-10

^mxreport 11-10

N

National language independence G-13
report on language translations 12-44

National Language Reports utility 12-44
sample report 12-45

Network G-13
Configuration G-13
DT Network 1-13
M/NET 1-13

%new 11-3

%newext 11-3

{%newpage} 11-6

NEXT subroutine 11-25

%NOCHECK 9-3, 9-44

Node G-13

Non-cursor-based SQL 8-2, G-13
passing information into M variables 8-3
using the INTO clause 9-18

Non-database forms G-13

#noshow 5-14

NOT IN operator 9-32

O

Object code 1-11, G-14

Object code routines 1-18, 5-2
extensions 6-3
maintaining backup versions 6-5

Object Compilation Driver Results window 12-21
viewing error messages 12-22

Object Compile Driver Items window 12-12
advanced options for forms 12-15, 12-16
advanced options for routines 12-18
fields defined 12-13

Object Compile Driver utility 12-5, G-14
advanced options 12-14
compilation error messages 12-21
compiling a configuration 12-20
compiling a configuration from within an M program 12-22
defining the contents of a compilation configuration 12-12
editing object definitions 12-19
setting defaults for compilation options 12-8
using 12-5
viewing the results of a compilation 12-21

Object Compile Driver window 12-7
fields defined 12-7

Object definition globals 11-9
list of 11-9
located in common directory 11-11

Object definitions
checking integrity of 12-23
looking up routine prefix for 12-42
searching for strings in 12-35

Object Integrity Checker utility 12-23, G-14
automatic error fixing 12-34
checks made on base tables 12-24
checks made on forms 12-25
checks made on menu objects 12-26
checks made on reports 12-25
checks made on triggers 12-27
checks made on views 12-24
enabling error fixing mode 12-30
running 12-29
sample report 12-33

Object Routine Prefix utility 12-42
sample display 12-43
using 12-42

Object String Search utility 12-35
sample report 12-39
using 12-35

Objects G-14
stored in globals 11-9

%ok 11-2

%old 11-3

%oldext 11-3

^%omc 11-12

^%oms 11-12

One-way outer joins 2-14, 9-5, G-15

Open M G-14
Relational environment G-17

Open M/SQL 1-2, G-15
accessing the relational database 2-6
application development strategies 1-4
applications portable across M systems 1-14
common directory 11-11
compiler 1-11
defining a database 1-10
development environment 1-10, 3-4
distributed data processing 1-19
extensions to relational model 2-14
global database 1-9, 1-16
globals 11-9
hand-coded programming 1-11
hardware and operating system environments 1-8
implementation of SQL 9-1
in a vendor-independent host M environment 1-14
integration of M and SQL 1-15
keyboard mapping for supported terminal types D-1
list of reserved words B-1
list of supported terminal types C-1
M language processor 1-9
memory environment 1-9
networking 1-13
open systems architecture 1-2
percent variables 11-2
program structure 1-17
programming methods 3-2
query optimizer 1-12
Relational Data Dictionary 2-5
relational database 2-1, G-15
schemas 2-4
SQL language processor 1-9
supported M systems 1-14
system-generated routines 11-43

Open M/SQL Developer 1-3

Open M/SQL for DSM 1-14
terminal types supported for C-3
Open M/SQL for DTM 1-14
terminal types supported for C-3
Open M/SQL for MSM 1-14
terminal types supported for C-3
Open M/SQL Main Menu 12-2
OPEN statement 8-4
Open systems 1-2
ORDER BY clause 10-8
Override lookup queries
inserting code 11-22

P

{%pagenum} 11-6
Parent ID G-15
Parent tables 2-17, 9-9
{%parent_reference} 11-3, 11-20
Percent globals 11-12
list of 11-12
Percent variables 11-2
enclosed in curly braces 11-2
list of 11-2
referencing within triggers 11-19
return prefix 11-20
Plus collation sequence function 9-39
Pop-up menus 1-12, G-15
cascading G-2
pulled down from menu bar G-16
popvars^%msqlutl function 11-41
Preprocessor commands 5-4
#; 5-15
#define 5-7
#else 5-11
#elseif 5-11
#endif 5-11
#if 5-11
#ifdef 5-11
#ifundef 5-11
#include 5-13
#noshow 5-14
#show 5-14
summary of 5-7
#undef 5-9
#undefine 5-7
Preprocessor functions 5-5
##sql 5-16, 8-2
&sql 5-16, 8-2
summary of 5-16
##vendor 5-17
{%presave} 11-4, 11-20
Primary key 2-3, G-15
Privilege operators 9-47
Privileges G-16
for running queries 10-29
granting 9-47
revoking 9-49
summary of 9-49
suppressing checks 9-50
Processes G-16
Product Activation Key G-16
Programmer mode G-16
Project operation 2-9
Prompt G-16
_PUBLIC (UserName) 9-48
Pull-down menus G-16
pushvars^%msqlutl function 11-40

Q

^%qarmisc 11-12
Queries 2-6, 10-1, G-16
ALTER VIEW 9-57, 10-17
ASCII-delimited output format 10-14, 10-26, 10-35
%CHECKPRIV 9-50, 10-17
compiled routines of 11-47
copying 10-36
CREATE VIEW 9-57, 10-17
creating via the Interactive Query Editor 10-18
creating via the Query Generator 10-5
DELETE 10-17
DROP VIEW 9-58, 10-17

embedded in M macro source code 8-1, 9-17
formatting output for import by other products 10-35
generation facilities 10-2
GRANT 9-47, 10-17
INSERT 10-17
invoking from M code 11-31
listing (detailed) 10-40
listing (short) 10-39
privileges needed to run 10-29
query owner 10-14, 10-26
REVOKE 9-49, 10-17
routine prefix of 10-14, 10-26
running 10-29
running from M code 10-31
run-time measure 10-14, 10-26
SELECT 10-18
selecting an output device 10-32
storage globals 11-47
UPDATE 10-18

%QUERY 9-3

Query call syntax (from M) 10-31, 11-31
parameters defined 10-31, 11-31

Query Definition Advanced Features window 10-13
fields described 10-14

Query Definition window 10-7
advanced features option 10-13
fields described 10-7
menu bar options described 10-11

Query Editor 10-17, G-9
accessing 10-2
compiling queries 10-28
copying queries 10-36
creating queries 10-18
editing commands 10-23
horizontal options menu 10-23
running queries 10-28, 10-30
screen display 10-21
supported query types 10-17
using on-line help 10-24

Query Editor Advanced Options window 10-25
fields described 10-26

Query Editor horizontal options menu 10-23
options described 10-23

Query Generator 10-5, G-17
accessing 10-2
compiling queries 10-16
copying queries 10-36
creating SELECT-statement queries 10-5
displaying list of fields from tables named in FROM clause 10-11
running queries 10-16, 10-29

Query optimizer 1-12

Query-based views G-17
altering 9-57
creating 9-57
deleting 9-58
naming in FROM clause 9-58
restrictions on defining 9-58

Querying the database 2-6

R

^%rakey 11-12

^%rakeys 11-12

^%RDE 11-12

^%rde 11-12

$$read^%msql function 11-36
parameters defined 11-37

REFERENCES privilege 9-49

Referential integrity G-17

^%rekey 11-12

Relation G-17

Relational Client G-18

Relational Data Dictionary 2-5, 3-4
accessing data from 2-6

Relational database G-17
accessing data from 2-6
characteristics 2-2
fields 2-2
First Normal Form 2-2
for non-ISM systems 2-4
overview 2-1
querying via SQL 2-6

structure of 2-2
tables 2-3

Relational environment G-17

Relational Gateway G-17

Relational model 2-1, G-18
First Normal Form 2-2
implicit joins 2-15, 9-6
InterSystems extensions to 2-14
one-way outer joins 2-14, 9-5
schemas 2-4
support for multi-line fields 2-21

Relational operations 2-7
cartesian product 2-7
joins 2-11, 9-5
project 2-9
restrict 2-10

Relational Server 1-3, G-18

Remote directory syntax (for routines) 6-8
restrictions 6-9

%REPORT 9-3

Report call syntax (from M) 11-30
parameters defined 11-30

Report triggers 11-18
action types 11-18

{%report-end} 11-7

Reports 1-5, 1-12, G-18
checks made by Object Integrity Checker utility 12-25
compiled routines of 11-47
Detailed Query Listing report 10-40
invoking from M code 11-30
List Queries report 10-39
storage globals 11-47
triggers 11-18

Required fields G-18
conditionally required fields 11-24

Reserved words B-1

Restrict operation 2-10

{%retrieved} 11-6, 11-20

Return prefix percent variables 11-20

{%return_action} 11-5, 11-20

{%return_filetype} 11-5, 11-20

{%return_presave} 11-5, 11-20

{%return_savedata} 11-5, 11-20

{%return_timeout} 11-5, 11-20

REVOKE statement 9-49, 10-17
CASCADE option 9-50
GRANT OPTION FOR keyword 9-50

ROLLBACK 9-45

^ROUTINE 11-9

Routine Line Editor 3-3, 4-2

Routine management utilities 3-3, 7-1
for non-ISM systems 1-18
invoking from Developer Utilities menu 7-3
invoking from M programmer prompt 7-3
invoking from the Developer Utilities menu 12-40
summary of 7-2

Routine prefix
calling forms by routine prefix 11-28
calling menu objects by routine prefix 11-33
default names for all objects 11-43
using the Object Routine Prefix utility to find origin of 12-42

Routine prefix (for queries)
created via the Query Editor 10-26
created via the Query Generator 10-14

Routine sets 6-10
creating 6-10
using 6-10

Routines 1-17
automatic syntax checking 4-23
compiling 6-11, 7-17
converting intermediate code to macro source code 6-2
copying 7-19
creating 6-2
creating in the FSE 4-7
deleting 6-14, 7-27
deleting backup versions 6-14, 7-29
extensions 6-3

generated for base tables 11-44
generated for forms 11-46
generated for menu objects 11-46
generated for queries 11-47
generated for reports 11-47
intermediate code routines 1-17
keeping routine levels synchronized 6-15
listing routines in current directory 7-11
load from file 7-9
loading automatically into FSE 4-6
loading into the FSE 4-6
macro source routines 1-17, 5-1, 5-2, 6-2
maintaining backup versions 6-5
management utilities 1-18, 7-1
names of system-generated routines 11-43
naming 6-3
object code routines 1-18
output to file 7-7
producing backup versions 6-12
referencing routines in other directories/ computers 6-8
restoring backup versions 6-13
routine sets 6-10
search for all in a set of text strings 7-25
search for and replace text strings 7-15
search for one in a set of text strings 7-23
set # of backup versions 7-28
shuffling and renumbering backup versions 6-12
using wildcard symbols to specify 6-6
utility for selecting routines 7-30
writing 6-2

Row selection G-19

%ROWCOUNT 9-3

RowID 2-3, G-18

Rows 2-2, G-18

Run Existing Queries utility 10-30

S

{%savedata} 11-4, 11-20

Schema 2-4

SELECT clause 10-7

SELECT list item G-19

SELECT privilege 9-49

SELECT queries 10-18
ASCII-delimited output format 10-35
creating via Query Generator 10-5
handling multi-line fields 9-24
in non-cursor-based embedded SQL 8-2
passing information into M arrays 9-19
selecting an output device 10-32
template for defining 10-7

Server G-19

Server masters G-19

Server processes G-19

setaid^%msql function 11-39

Short help messages G-19

#show 5-14

Sign-on G-19

Single-row forms G-20

Space collation sequence function 9-39

SQL 1-2, 1-15, 2-6, G-20
accepts multi-line fields 2-21
accessing the relational database 1-16
aggregate function extensions 9-11
collating output values 9-37
creating queries via the Interactive Query Editor 10-18
creating queries via the Query Generator 10-5
cursor-based 1-15, 8-4, G-4
embedded in M macro source code 1-15, 8-1, 9-17
extensions to SQL operators 9-32
extensions, summary of 9-2
handling duplicate rows 9-16
handling internal and external values for fields 9-26, 9-29
handling multi-line fields 9-24
handling of subqueries 9-52
inserting SQL code into triggers 11-21
InterSystems extensions to 2-14
joins 9-5
keyword extensions 9-3
language processor in Open M/SQL 1-9
list of error messages A-1
list of reserved words B-1

non-cursor-based 1-15, 8-2, G-13
privilege operators 9-47
programming methods 3-2
query generation facilities 10-2
querying the relational database 2-6
relational operations 2-7
summary of supported M operators 9-30
symbol extensions 9-4
transaction processing 9-45
using the INTO clause 9-18

##sql 5-16, 8-2

&sql 5-5, 5-16, 8-2

SQL code
inserting into applications 11-13

SQL Menu 10-3
options described 10-4

SQL operators
Concatenation operator 9-32
LIKE predicate 9-34
NOT IN operator 9-32
%STARTSWITH 9-33

SQLCODE variable 9-51, A-1

Stand-alone captions G-20

%STARTSWITH 9-3, 9-33

Subqueries 9-52, G-20
extensions to 9-55
FOR ALL operator 9-56
FOR SOME operator 9-56
syntax 9-52
using in FROM clauses 9-54
using in SELECT clauses 9-55
using in WHERE clauses 9-52

Symbols (in SQL) 9-4

^%sys 11-12

System manager's directory G-20

T

Tables 2-2
see also base tables, child table, parent tables, virtual tables, views

^%task 11-12

Terminal types
in Open M/SQL for DSM environment C-3
in Open M/SQL for DTM environment C-3
in Open M/SQL for MSM environment C-3
list of supported terminal types C-1

%THRESHOLD 9-3

{%time} 11-6

{%timeout} 11-4, 11-20

Transaction G-20

Transaction processing 9-45
%BEGTRANS 9-45
%INTRANS 9-45

Translations (for system-generated messages and menus)
reports on 12-44

Triggers 1-5, G-20
action types 11-16
base table 11-16
checks made by Object Integrity Checker utility 12-27
form 11-17
inserting M code 11-16, 11-21
inserting SQL code 11-16, 11-21
referencing fields 11-19
referencing Open M/SQL percent variables 11-19
report 11-18

U

#undef 5-7, 5-9

UPDATE privilege 9-49

UPDATE queries 10-18
handling multi-line fields 9-25
in non-cursor-based SQL 8-3
passing information into M arrays 9-22
with VALUES clause 9-23

%UPPER 9-3, 9-41

UPPER 9-38

%urchange 7-15
compiling routines 6-11
producing backup versions 6-12
%urcomp 6-11, 7-17
producing backup versions 6-12
%urcopy 7-19
compiling routines 6-11
producing backup versions 6-12
restoring backup versions 6-13
%urdel 6-14, 7-27
%urdir 7-11
long form display 7-14
short form display 7-12, 7-13
%urfand 7-25
%urfind 7-23
%urload 7-9
compiling routines 7-10
load options 7-9
%urprint 7-7
output to file 7-8
output to printer 7-8
output to screen 7-7
%urpurge 6-14, 7-29
%urset 7-30
parameters described 7-31
%urverma 6-12, 7-28
UserName
establishing from M code 11-39
stored in %msql local variable 11-39
Utilities
Copy Query 10-36
Detailed Query Listing 10-40
List Queries 10-39
%msqlutl 11-40
National Language Reports 12-44
Object Compile Driver 12-5, G-14
Object Integrity Checker 12-23, G-14
Object Routine Prefix 12-42
Object String Search 12-35
program and object management 3-4
routine management 3-3, 7-2
Run Existing Queries 10-30
^UTILITY 11-9

V

%val 11-2
Validation code 11-24, G-21
VALUES clause 8-3, 9-23
Variable window placement G-21
Variables G-21
displaying list of critical variables 11-42
list of percent variables 11-2
naming conventions 11-2
percent (%) variables 11-2
pushing onto a stack 11-40
referencing within inserted code 11-15
referencing within triggers 11-19
reinstating from stack 11-41
##vendor 5-17
Versions (for routines) 6-5
Vertical menus 1-12, G-21
Views 1-10, G-21
checks made by Object Integrity Checker utility 12-24
query-based 9-57, G-17
Virtual fields G-21
Virtual tables 1-10, 2-3, G-21

W

WHERE clause 10-8
Wildcard symbols
for routine extensions 6-7
for routine names 6-6
for version numbers 6-7
windcln^%msql function 11-38
Window cleanup function 11-38
Windows G-22
erasing from screen 11-38
Windows (in the Full Screen Editor) 4-18
Word-processing fields G-22
write^%msql function 11-35
parameters defined 11-36