MAX/REXX

MAX/REXX V3.4.0

User Reference Manual

MAX SOFTWARE™
Denver, Colorado
MAXimizing your performance

CA, INC.

Contact Information

Corporate Headquarters

CA, Inc.
One CA Plaza
Islandia, NY 11749
USA

www.ca.com

Technical Support

For online technical assistance and a complete list of locations, primary service hours, and telephone numbers, contact Technical Support at http://ca.com/support

Web Site

http://ca.com/support

Disclaimer

Disclaimer of Warranties and Limitation of Liabilities

The staff of MAX Software has taken due care in preparing this manual; however, nothing contained herein modifies or alters in any way the standard terms and conditions of the MAX Software, Inc. standard software evaluation agreement; purchase agreement; lease agreement; or rental agreement by which this software was acquired; nor increases in any way MAX Software's liability to the customer. In no event shall MAX Software be liable for incidental or consequential damages in connection with or arising from the use of this manual or any program contained herein.

Release 3.4.0 (November 2005)

MVS, DB2, IMS, REXX, TSO/E, and ISPF are software products of International Business Machines Corporation.

MAX MVS/UTIL (a complete set of data file manipulation tools with the following 3 components: MAX Data/Util, MAX/PDF and MAX/Batch); MAX IMS/UTIL (a complete set of IMS database manipulation tools with the following 2 components: MAX/IMS Online and MAX/IMS Batch); MAX DB2/ UTIL (a complete set of DB2 database manipulation tools); and MAX/REXX (an interface between REXX and VSAM, SAM, PDS and DB2 data) are trademarks of MAX Software, Inc.

TABLE OF CONTENTS

LIST OF FIGURES

REVISIONS

Release 3.4.0: November 2005

Release 3.3.0: March 2004

- Additional features added to RXSQL, including: **CODEPAGE** support for **SELECT INTO FILE**.
- Additional features added to RXMVS, including:

CONVERT	**CPCLASS**	**CPLIST**	**CPNAME**
DECODE64	**ENCODE64**	**ISALNUM**	**ISALPHA**
ISBLANK	**ISCNTRL**	**ISDIGIT**	**ISGRAPH**
ISLOWER	**ISPRINT**	**ISPUNCT**	**ISSPACE**
ISUPPER	**ISXDIGIT**	**TOLOWER**	**TOUPPER**

Release 3.2.0: March 2003

- Improvements to RXSQL, including: SELECT INTO FILE and EXECUTE USING FILE.

Release 3.1.0: June 2002

- New products added to the release: MAX DB2/UTIL.
- Improvements to RXSQL, including:
  1. DB2 V6 data types: `DECIMAL(31)`, `ROWID`, `BLOB`, `CLOB` and `DBCLOB`.
  2. **DECLARE CURSOR . . . WITH HOLD**.
  3. **DECLARE GLOBAL TEMPORARY TABLE** statement.
  4. **DESCRIBE INPUT** statement.
  5. **SET** `:hostvar = special register` statement.
  6. Indicator variables in **INTO** clause and **NULL** keyword in **USING** clause.
  7. Reduced virtual storage usage.
  8. Improved floating point support.

Release 2.5.0: July 2001

- Support for files with multiple record layouts using mapping criteria.

Release 2.4.0: September 2000

- New products added to the release: MAX IMS/UTIL.

Release 1.5.3: November 1998

- Additional features added to RXVSAM.
- Additional features added RXMVS:

ABEND	**ALLOCATE**	**CALCGDIF**	**CONNECT**
CALCJDIF	**DISONN**	**FREE**	**LIST**
LISTCAT	**LISTVTOC**		

Release 1.3.0: January 1997

- Additional features added for PDS processing: RXVSAM several additional commands **DIR**, **FIND**, **ADDMEM**, **REPLMEM**, **DELMEM**, **RENAME**.
- RXMVS features added:
  1. Sort array of variables.
  2. Perform various date calculations.
  3. **ENQ**ue and **DEQ**ue on resources.
  4. **LINK** to other non-REXX programs.
  5. Dynamically assigns variable names.
  6. Sequential update capability improved.

Release 1.2.0: March 1995

- PL/I copybook support provided.
- **WHERE** keyword record selection logic added for SAM and VSAM.
- Compile feature added.
- Performance enhancements that significantly improve the performance of the SQL, SAM, VSAM, and Compiler features.

Release 1.1.0: March 1994

- Initial release.

PREFACE

This book provides a guide and reference to the various functions of MAX/REXX for MVS. Use this book to learn and use the MAX/REXX product.

It describes:

- An introduction to the product.
- Guideline information for coding statements.
- SAM/VSAM/PDS statement syntax.
- SAM/VSAM/PDS Special Variables and return code information.
- SQL statement syntax.
- SQL Special Variables and return code information.
- MVS Commands with statement syntax.
- MVS Special Variables and return code information.
- A sample REXX VSAM application.
- A sample REXX SQL application.
- Compile sample JCL and syntax.

Who Should Read This Book

This book is for programmers, database administrators, system programmers, or other technical persons who need access to DB2, SAM, VSAM or PDS data sets using the REXX language, want to utilize the MVS features, or want to compile REXX programs. Users are expected to be familiar with REXX programming.

Notational Conventions

The following notational conventions are used in this manual:

- Uppercase commands and their operand(s) should be entered as shown but need not be in uppercase.
- Operand(s) shown in lower case are variables and a value should be substituted for them.
- Operand(s) shown in brackets [ ] are optional, with choices indicated by a vertical bar |. One or none may be chosen; the defaults are underscored.
- Operand(s) shown in braces { } are alternatives; one must be chosen.
- An ellipsis (...) indicates that the parameter shown may be repeated to specify additional items of the same category.

How This Book is Organized

This book contains the following chapters:

Chapter 1: Introduction
describes the need and overall benefits and uses for MAX/REXX for MVS.

Chapter 2: General Concepts
describes an overview of how the MAX/REXX product functions.

Chapter 3: SAM/VSAM/PDS Processing
describes the various commands and operand(s) which may be invoked.

Chapter 4: SQL Processing
describes the various commands and operand(s) which may be invoked.

Chapter 5: MVS Processing
describes the various commands and operand(s) which may be invoked.

Chapter 6: Text File Processing
[insert description here]

Chapter 7: Interpretive Compiler Feature
describes the MAX/REXX interpretive compiler options.

Appendix A: RXVSAM Return Codes
lists the various return codes that may be returned, and their meaning and where to find additional information about the return code.

Appendix B: RXSQL Return Codes
lists the various return codes that may be returned with their meaning. It also indicates where to find additional information about the return codes.

Appendix C: RXMVS Return Codes
lists the various return codes that may be returned from RXMVS along with their meanings. It also indicates where to find additional information about the return codes.

CHAPTER 1: INTRODUCTION

REXX Overview

REXX is IBM's standard procedural language and is distributed as part of the current release of most IBM SAA compliant operating systems.

Rexx is an Interpretive Language

As an interpretive language, REXX allows a developer to switch from entering the program to executing the program without any intervening (preprocessing, compiling, link-editing, binding) steps. This benefit allows programs to be developed and tested very quickly. Following the testing phase, the program may then be compiled.

Rexx requires no data declaration statements

All fields (otherwise known as variables) are internally processed based upon their content. A variable that contains a valid number is considered numeric, and arithmetic operations may be performed on it. Otherwise, it is considered non-numeric. This feature significantly reduces the time and coding effort for developing a REXX program.

The Rexx Language is easy to learn and use

The REXX language has free format, non-restrictive data naming and uses minimal punctuation. It supports the major programming constructs (**IF-THEN-ELSE**, **DO** loops, **SELECT**, subroutine calls, array processing) needed for powerful development. REXX is easy to learn and use, yet extensive enough to build commercial strength interactive or batch applications

The Need for MAX/REXX

MVS REXX is powerful yet easy to use and programmers love it. However, it has no capabilities for accessing:

- DB2
- Partitioned Data Sets
- VSAM data
- Individual fields within files

In addition, there are many features of MVS that are not accessible by the basic REXX language.

Once development of a program is complete, there is a need to compile the program to improve performance and provide the same type of security for the source as other languages.

These are the principle reasons for the development of the MAX/REXX product:

- MAX/REXX allows the REXX language to be used to solve business problems.
- MAX/REXX provides the capabilities to process business data with the performance needed in a commercial strength language.

Introduction to the SAM/VSAM/PDS Feature

MAX/REXX allows REXX programs to have full access to all types of SAM and VSAM files. In addition, PDS file access is also supported.

MAX/REXX applications may be tested immediately without any intervening steps. The programmer need not be concerned with the conversion of internal data formats because MAX/REXX handles all of this automatically. There is minimal or no training required because MAX/REXX uses command level syntax with which MVS and CICS programmers are already familiar.

Unique Use of COBOL/PL/I Layouts

MAX/REXX allows an externally defined COBOL or PL/I layout to be used. This layout is used by MAX/REXX to handle the stringent requirement for data typing needed for an individual record in a SAM or VSAM file and REXX variables which have no data-typing. After any record level access, REXX variables are available for any field in the file. All data conversions to and from the file layout and the REXX variables are handled automatically.

Without this feature, each REXX program would need additional logic added to extract out individual fields from the record. Logic would be needed to edit individual fields (for example, to insert a decimal point or to convert external decimal data to display format).

Mapping of Multiple Record Layout Files

MAX/REXX supports files containing mutiple record layouts with a feature referred to as mapping criteria. Mapping criteria relates a copybook to a record type. Mapping criteria is built using the MAX Data/Util product. If a file is opened with mapping criteria, each record will be returned in the field name variables for the appropriate record layout that matches that record.

Separation of Data Conversions from Applications Logic

With MAX/REXX, programmers focus on the application problem and not data conversion. This unique ability of MAX/REXX dramatically reduces the development and maintenance efforts for sequential and VSAM applications. Programs are smaller, easier to read and maintain, and a file layout change need not require any change to the REXX program itself.

Powerful Record Selection Criteria

Often it is not necessary to process all of the records within a file. The record selection feature of MAX/REXX allows only those required records to be returned to the REXX program.

By passing selection criteria with the **READNEXT** or **READPREV** command, RXVSAM will return only those records that meet the criteria. This dramatically improves the performance and allows REXX programs to process large volumes of data efficiently.

Sample SAM/VSAM Update Program

The following COBOL copybook and REXX program demonstrate a selective update to a data file in the sequential mode. The REXX variables **AMOUNT** and **STATE** correspond to the field names from the supplied COBOL file layout.

Any field defined in the file layout is available as a REXX variable. All conversions to and from the file layout and the REXX variables are handled automatically.

```
*    THIS IS A SAMPLE COBOL COPYBOOK
          01  EMPLOYEE-RECORD.
              05  EMPLOYEE-ID.
                  10 RECORD-TYPE     PIC X(1).
                  10 EMPLOYEE-CODE   PIC 9(6).
              05  EMPLOYEE-NAME.
                  10 NAME-FIRST      PIC X(9).
                  10 NAME-LAST       PIC X(15).
                  10 NAME-MIDDLE-I   PIC X.
              05  EMPLOYEE-ADDRESS.
                  10 STREET-ADDR     PIC X(20).
                  10 CITY            PIC X(10).
                  10 STATE           PIC XX.
              05  AMOUNT             PIC 9(9)V99 COMP-3.
```

Figure 1: Sample Copybook panel

Sample VSAM REXX Program

The following sample REXX program displays how to sequentially update a file.

```
/* REXX ***************************************************************** */
/* DOC: Sample SAM/VSAM sequential update program                        */
/* ***************************************************************** */

RC=RXVSAM("OPEN FILE(FILE1),          /* open for sequential update  */
          COPYBOOK('SAMPLE.VSAM.COPYLIB(LAYOUT1)') SEQUPD")
                                      /* repeat until end of file    */
DO WHILE RXVSAM("READNEXT FILE(FILE1)")=0
   IF STATE="CO" THEN DO              /* select specific records     */
      AMOUNT=AMOUNT*1.15              /* add 15% to amount           */
      AMOUNT=AMOUNT+.0050             /* round the amount            */
      AMOUNT=TRUNC(AMOUNT,2)          /* truncate to 2 decimal digits*/
      RC=RXVSAM("REWRITE FILE(FILE1)")
   END                                /*                             */
END

IF VSAMCODE>8 THEN DO                 /* if error - display message  */
   SAY 'VSAMCODE='VSAMCODE 'VSAMMSG='VSAMMSG
END

RC=RXVSAM("CLOSE FILE(FILE1)")

EXIT 0
```

Figure 2: Sample VSAM REXX Program panel

Introduction to the SQL Feature

The MAX/REXX SQL product extends the REXX language by allowing SQL statements to be coded directly in-line within the REXX program.

MAX/REXX SQL applications may be tested immediately without any intervening steps. There is minimal or no training required because MAX/REXX SQL supports the full SQL syntax as used in other languages like COBOL and PL/I. MAX/REXX SQL provides immediate and extensive feedback on any error condition.

The MAX/REXX SQL Advantage

MAX/REXX SQL supports row at a time processing (`CURSOR` based statements) that other interpretive solutions, such as QMF and SPUFI, do not support. Products that do not support `CURSOR` statements require either extensive memory, or additional overhead to store the query results into an intermediate file. The results of a query on a large DB2 table may not fit into memory and writing it to a temporary intermediate file is very inefficient and time-consuming. With MAX/REXX SQL, a row at a time of data is moved from DB2 and stored directly into REXX variables for processing.

MAX/REXX SQL programming is straightforward and concise. All data conversions to/from DB2 columns and REXX variables are handled automatically. You get the efficiency of static SQL statements (like other compiled languages) along with the conciseness and interpretive benefits of REXX.

```
/* REXX  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* DOC: Lists tables defined to a specific DB2 subsystem                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
                                        /* connect to DB2                 */
   IF RXSQL("CONNECT DB2P")<>0 THEN CALL CLEANUP                          */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* List name, dbname, tsname from SYSIBM.SYSTABLES                        */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
   IF RXSQL(" DECLARE C1 CURSOR FOR",
     "SELECT NAME, DBNAME, TSNAME FROM SYSIBM.SYSTABLES",
     "ORDER BY DBNAME, TSNAME, NAME")<>0 THEN CALL CLEANUP
                                        /*  open the cursor               */
   IF RXSQL("OPEN C1")<>0 THEN CALL CLEANUP
   DO WHILE RXSQL("FETCH C1")=0         /* fetch each row in table*/
     SAY 'DBNAME='SUBSTR(DBNAME,1,12),  /* list columns                   */
         'TSNAME='SUBSTR(TSNAME,1,12),  /*                               */
         'NAME='NAME                    /*                               */
   END                                  /*                               */
   IF SQLCODE<>100 THEN CALL CLEANUP    /* test for end of file          */
                                        /*                               */
   RC=RXSQL("CLOSE C1")                 /*                               */
   CALL CLEANUP                         /*                               */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Clean up and EXIT                                                      */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
CLEANUP:
IF SQLCODE<>0 THEN                    /*                          */
      SAY 'RXSQL failed RC='SQLCODE SQLMSG /*                          */
                                        /*                               */
   CALL "RXSQL" "DISCONNECT"            /* disconnect from DB2           */
                                        /*                               */
   EXIT 0                               /*                               */
```

Figure 3: Sample SQL Program panel

Introduction to the MVS Feature

The MAX/REXX product further enhances the REXX language by allowing the programmer access to features of MVS not supported by the REXX language itself, as well as other common processing routines.

As with the other features of MAX/REXX, the applications may be tested immediately after being developed. Features supported under the MVS component of the product bring heightened capability to the application developer, and give increased control over applications.

Added Capability with the MVS Feature

- The ability to **ENQ** and **DEQ** allows REXX applications to serialize on critical resources.
- Extensive date manipulation is provided with this component, including both the conversion of and calculation of dates. Dates are supported in full century, year, month, and day format.
- Internal **SORT** capabilities allow the application to sort an array of variables in any sequence with the single execution of a command.
- Program linkage is available using the **LINK** command. This allows for execution of non-REXX programs though the use of traditional MVS load module search sequence and passed parameter linkage.
- Finally, the **ASSIGN** command allows for the dynamic assignment of a variable name without the necessity of the **INTERPRET** statement. The **INTERPRET** statement is not supported by the REXX compiler.

Sample REXX Program using the MVS Feature

The following sample REXX program utilizes the MVS feature of the MAX/REXX product. A partitioned data set's directory is loaded into a variable array. The variable array is then sorted in descending sequence, using columns 1 through 8. This particular example could be appropriate when concatenated files are being processed.

```
/* REXX  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* DOC: RXVSAM/RXMVS - Sort a concatenated PDS by member name        */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
                                       /*                           */
  "ALLOC FI(DD1) DA('SYS1.CLIST' 'SYS1.CMDLIB') SHR"
  IF RC<>0 THEN DO                     /*                           */
    SAY 'RC='RC 'ALLOCATE FAILED'      /*                           */
    CALL CLEANUP                       /*                           */
  END                                  /*                           */
                                       /*                           */
  IF RXVSAM('OPEN FILE(DD1) INPUT')<>0 THEN DO
    CALL CLEANUP                       /*                           */
  END                                  /*                           */
                                       /*                           */
  CALL RXVSAM 'DIR FILE(DD1) INTO(INPUT.)' /* Fetch directory        */
  CALL RXVSAM 'CLOSE FILE(DD1)'        /* Close file                */
                                       /*                           */
 "FREE FI("DD1")"                      /* Free file                 */
                                       /*                           */
 CALL RXMVS 'SORT FROM(INPUT.) INTO(OUTPUT.) START(1) LENGTH(8) SEQ(D)'
                                       /*                           */
  DO I=1 TO OUTPUT.0                   /* Loop - end directory      */
    SAY 'SEQ='I SUBSTR(OUTPUT.I,1,8)   /*  Say member name          */
  END                                  /*                           */
                                       /*                           */
EXIT 0                                 /*                           */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Clean up and exit                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
CLEANUP:
                                       /*                           */
   IF VSAMCODE<>0 THEN DO              /*                           */
      SAY 'RXVSAM failed RC='VSAMCODE VSAMMSG /*                    */
   END                                 /*                           */
                                       /*                           */
   IF MVSCODE<>0 THEN DO               /*                           */
      SAY 'RXMVS failed RC='MVSCODE MVSMSG /*                       */
   END                                 /*                           */
                                       /*                           */
   EXIT 0                              /*                           */
```

Figure 4: Sample Program using MVS Feature

Introduction to the Interpretive Compiler Feature (I-Compiler)

The MAX/REXX I-Compiler compiles REXX source programs into executable object modules. The compiled programs may be executed directly from JCL, called from a program, or invoked as a TSO command procedure. This compiler provides the same security and change control as other languages such as COBOL or PL/I. There may be some improved overall performance derived from pre-parsing, compressed source, and improved program load. However, performance of the execution of a single REXX instruction is not improved.

CHAPTER 2: GENERAL CONCEPTS

RXSQL, RXVSAM, and RXMVS Modules

There are three (3) main modules (RXSQL, RXVSAM, RXMVS) for invoking MAX/REXX services.

1. RXSQL processes SQL statements for accessing DB2.
2. RXVSAM processes command level statements for processing SAM, VSAM and PDS files.
3. RXMVS processes commands commonly needed but not found in the REXX MVS language.

Placement of Statements

MAX/REXX statements may be placed anywhere in the REXX program that any other REXX instruction may be placed. MAX/REXX statements are processed as they are encountered just as other REXX instructions.

For example, MAX/REXX statements may be placed within a **DO** loop and each time through the loop; the MAX/REXX statement will be processed. Place MAX/REXX statements as they are needed in the logic of the program.

How to Invoke MAX/REXX Modules

There are three (3) methods to invoke MAX/REXX modules and process statements:

1. Use a host command:
   ADDRESS RXVSAM, ADDRESS RXSQL, ADDRESS RXMVS
2. Use a **CALL** to an external routine:
   CALL "RXVSAM", CALL "RXSQL", CALL "RXMVS"
3. Use the function calls:
   RXVSAM(...), RXSQL(...), or RXMVS(...)

Within a REXX program any one, or combination, of the three methods for invoking MAX/REXX modules may be used. There are different reasons why a user may choose one method over another for a specific request. The following summaries explain the differences of invoking MAX/REXX modules using the three different methods.

Host Command Method

ADDRESS RXVSAM, ADDRESS RXSQL, ADDRESS RXMVS

Upon completion, a variable named `RC` contains the return code.

The "`CALL ON ERROR`" or "`SIGNAL ON ERROR`" routine is automatically invoked for non-zero return codes. This can facilitate the coding of a common error processing routine.

`TRACE COMMANDS` or `TRACE ERROR` can be used to limit tracing.

Call External Routine Method

CALL 'RXVSAM', CALL 'RXSQL', CALL 'RXMVS'

Upon completion, a variable named `RESULT` contains the return code.

`TRACE RESULTS` can be used to limit tracing.

Function Call Method

RXSQL(...), RXVSAM(...), or RXMVS(...):

The function call may be included as part of a REXX assignment clause.

For example: **MYRC=RXVSAM(...)**.

This may be used to name the return code variable.

The function call may be included as part of a REXX instruction clause.

For example: **IF RXSQL(...)<>0 THEN DO**.

This technique can be used to test the return code and execute the statement, reduce the size of a program and improve performance.

REXX and Host Command Environment

There is always at least one host command environment available to a REXX program. A “default” is always assigned by the system when the REXX program is invoked. Another host command environment may be temporarily selected, or the default changed by using the **ADDRESS** instruction. The name of the current default addressed host command environment may be queried via the **ADDRESS ( )** built-in function.

When the REXX language processes a clause that it does not recognize as a valid REXX instruction or assignment statement, then it will pass the resolved clause as a command to the currently addressed host command environment for processing.

To send a command to the currently addressed host command environment, simply specify any clause that is not a valid REXX instruction or assignment statement. The host command environment then processes the command and returns, setting a return code. The return code may be evaluated by using the special variable RC.

Before a command is routed to a host command environment for processing, the REXX language processor resolves (variable substitutions occur) the clause into a string. Commands may be dynamically prepared in this way.

To pass a static command (and avoid variable resolution), simply enclose the command in quotation marks (" ") or apostrophes (' ').

MAX/REXX Host Command Environments

In order to use the RXVSAM, RXSQL, or RXMVS host command environment, it must first be initialized.

The RXVSAM host command environment is automatically initialized when a file is opened. See the **OPEN** statement to initialize the RXVSAM host command environment.

The RXSQL host command environment is automatically initialized when the connection is established with a DB2 subsystem. See the RXSQL **CONNECT** statement to initialize the RXSQL host command environment.

The RXMVS host command environment is established on each invocation of RXMVS. The CONNECT and DISCONN commands are available to establish and retain the environment.

Assign Default Host Command Destination

When only the environment name is specified in an **ADDRESS** instruction it becomes the permanent default environment. Any clause that is not a valid REXX instruction or assignment statement will be passed to the default host command environment for processing.

Statements that use RXVSAM as the default host command environment should begin with 'RXVSAM'. The same is true for statements that use RXSQL or RXMVS as the default host command environment. These statements should begin with 'RXSQL' or 'RXMVS'. This will make programs that use RXVSAM, RXSQL, or RXMVS as a default host command environment easier to read and maintain.

```
. . .
                                  /* open for sequential processing */
  CALL "RXVSAM" "OPEN FILE(FILE1)
       COPYBOOK('SAMPLE.VSAM.COPYLIB(LAYOUT1)')"
. . .
  ADDRESS RXVSAM                  /* set the default environment    */
                                  /* issue RXVSAM statements        */
  "RXVSAM READNEXT FILE(FILE1)"
. . .
  CALL 'RXVSAM" "CLOSE FILE(FILE1)"
. . .
```

Figure 5: RXVSAM Example

```
. . .
                                  /* connect to DB2                 */
  CALL "RXSQL" "CONNECT DB2x"
. . .
  ADDRESS RXSQL                   /* set the default environment    */
                                  /* issue SQL statements           */
  "RXSQL REVOKE SELECT ON DSN8.TPROJ FROM PUBLIC
         WHERE DEPTNO = " DEPTNO
. . .
```

Figure 6: RXSQL Example

Temporarily Assign Host Command Destination

When a clause is specified along with the environment name in an **ADDRESS** instruction, the clause will be passed to the specified environment name for processing. However, the default host command environment is not changed. his is a method to temporarily override the default host command environment.

```
. . .
                                  /* open file update              */
  CALL "RXVSAM" "OPEN FILE(FILE1)
       COPYBOOK('SAMPLE.VSAM.COPYLIB(LAYOUT1)')
       UPDATE"
. . .                             /* pass statement to the
  ADDRESS RXVSAM ". . . statement" /* specified environment for     */
                                  /* processing                    */
. . .
  CALL "RXVSAM" "CLOSE FILE(FILE2)" /* disconnect by closing file   */
. . .
```

Figure 7: RXVSAM Example

```
. . .
                                  /* connect to DB2                */
  CALL "RXSQL" "CONNECT DB2x"
. . .                             /* pass statement to the
  ADDRESS RXSQL "SQL statement"   /* specified environment for     */
                                  /* processing                    */
. . .
  CALL "RXSQL" "DISCONNECT"       /* disconnect from DB2           */
. . .
```

Figure 8: RXSQL Example

```
. . .
                                  /* connect to RXMVS              */
  CALL "RXMVS" "CONNECT"
. . .                             /* pass statement to the
  ADDRESS RXMVS ". . . statement" /* specified environment for     */
                                  /* processing                    */
. . .
  CALL "RXMVS" "DISCONN"          /* disconnect from RXMVS         */
. . .
```

Figure 9: RXMVS Example

MAX/REXX External Routines

MAX/REXX modules RXSQL, RXVSAM, and RXMVS may be invoked as an external routine by issuing the REXX **CALL** instruction. External routine names should be capitalized and enclosed in quotation marks (" ") or apostrophes (' '). When enclosed in quotation marks or apostrophes, the search for internal routines is bypassed and performance may be slightly improved. The return code may be evaluated by using the special variable `RESULT`.

Note: Called external routines do not set the special variable `RC` as does a host command environment command.

```
CALL "RXVSAM"  "DELETE FILE(TEST02) RIDFLD(CUSTNO)"
CALL "RXSQL"  "DELETE FROM EMPLOYEE WHERE EMPNO = 00230"
CALL "RXMVS"  "DATE2JUL DATE("GREG_DATE") INTO(JUL_DATE)"
```

Figure 10: MAX/REXX External Routine

MAX/REXX Function Calls

MAX/REXX modules RXSQL, RXVSAM, and RXMVS may be invoked with a function call. A function call can be coded as part of a REXX assignment statement or as part of a REXX instruction. This technique can be used to reduce program size and improve performance.

```
DO WHILE RXSQL("FETCH C1")=0
. . .
END

or,

DO WHILE RXVSAM("READNEXT FILE(TEST1)")=0
. . .
END
```

Figure 11: MAX/REXX Function Calls

Process a Static Statement

When a clause is encountered which is enclosed in either quotation marks (" ") or apostrophes (' ') then it is passed directly for processing.

```
CUSTNO=10001                          /* identify a record         */
                                      /* delete the record         */
CALL "RXVSAM"  "DELETE FILE(TEST02) RIDFLD(CUSTNO)"
```

Figure 12: Static Statement Processing

In the above example, the `DELETE  FILE(TEST02)RIDFLD(CUSTNO)` is passed to RXVSAM for processing.

Dynamically Prepared Statement

When a clause is encountered that is not entirely enclosed in either quotation marks (" ") or apostrophes (' '), it is to be resolved (variables substitutions occur) first. The resolved clause is then passed for processing.

```
CUSTNO=10001                          /* identify a record         */
                                      /* build the delete statement*/
DELETE_STMT="DELETE FILE(TEST02) RIDFLD(CUSTNO)"
                                      /* delete the record         */
CALL "RXVSAM"  DELETE_STMT            /* passing prepared statement*/
```

Figure 13: Dynamically Prepared Statements

In this example, the `DELETE_STMT` variable is first resolved (variable substitution occurs). This occurs because the clause was not enclosed within quotation marks or apostrophes. The resolved character string `DELETE FILE(TEST02) RIDFLD(CUSTNO)` is passed to RXVSAM for processing.

Static and Dynamic Statement

A combination of static and dynamic coding may be used to prepare a statement. Enclose the static portion in either quotation marks (" ") or apostrophes (' ') and use variables in the portion of the statement that is to be dynamic.

```
FILE='TEST02'                       /* identify the file        */
CUSTNO=10001                        /* identify a record        */
                                    /* delete the record        */
CALL "RXVSAM"  "DELETE FILE("FILE") RIDFLD(CUSTNO)"
```

Figure 14: Combination Static/Dynamic Statement

Finally, in this example, the resolved character string `DELETE FILE(TEST02) RIDFLD(CUSTNO)` is passed to RXVSAM for processing.

Statement Continuation

A single statement may be placed on several lines. The normal rules for continuing a REXX statement apply with MAX/REXX statements. The maximum length of a literal string in REXX is 250 characters. If a statement exceeds 250 characters, a comma is needed to continue the statement.

```
"RXSQL CREATE TABLE DSN8.TEMPLD21
(EMP#         CHAR(6)
 LASTNAME     VARCHAR(15)
 FIRSTNAME    VARCHAR(12)",          /* continue statement  */
"PHONE        CHAR(4)
 JOBCD        DECIMAL(3)
 SALARY       DECIMAL(8,2) )"
```

Figure 15: Statement Continuation, example 1 of 3

Comments may be placed on every line of the statement if continuation is placed on every line as the following examples show.

```
RXSQL("INSERT TEMPLD21",              /* load a table           */
"SELECT EMPNO,",                      /* with all               */
"LASTNAME,",                          /* employees              */
"FIRSTNAME,",                         /*                        */
"PHONENO, JOBCODE, SALARY",           /*                        */
"FROM TEMPL",                         /* that work in           */
"WHERE WORKDEPT = 'D21'",             /* dept 21                */
"AND EMPNO NOT IN",                   /* and exclude the        */
"(SELECT MGRNO",                      /* managers               */
"FROM TDEPT",                         /* who work in            */
"WHERE DEPTNO = 'D21'")               /* dept 21                */
```

Figure 16: Statement Continuation, example 2 of 3

```
 CALL "RXVSAM"  "READNEXT",       /* read next record from       */
 "FILE(TEST01)",                  /* customer test file          */
 "INTO(RECORD)"                   /* into the RECORD variable    */
```

Figure 17: Statement Continuation, example 3 of 3

CHAPTER 3: SAM/VSAM/PDS PROCESSING

Summary

The following is a summary list of RXVSAM statements.

Open and Close Statements	
OPEN	Open a file.
CLOSE	Close a file.

Record Access and Positioning Statements	
DELETE	Delete a record.
READ	Read a record direct mode.
READNEXT	Read next record.
READPREV	Read previous record.
REWRITE	Update a record.
STARTBR	Start sequential processing.
ENDBR	End sequential processing.
WRITE	Add a record.

PDS Specific Access and Processing Statements	
DIR	Retrieve the directory information.
FIND	Position to a member within the PDS.
ADDMEM	Add a new member to a PDS.
REPLMEM	Replace a member of a PDS.
DELMEM	Delete a member of a PDS.
RENAME	Rename a member of a PDS.
REWRITE	Write a new record.

Field Access and Record Formatting Statements	
FORMAT FROM	Format variables from record.
FORMAT INTO	Format variables into record.
GETFIELD	Fetch a specific field from record.
PUTFIELD	Put a specific field in record.
SELECTCB	Select a copybook based upon record data.

Special Variables

VSAMCODE

The special variable **VSAMCODE** will always contain the return code following any invocation of RXVSAM. This variable is available whether RXVSAM was invoked via the host command environment (**ADDRESS RXVSAM**), an external routine (**CALL "RXVSAM"**), or as a function (**RXVSAM (...)**). This variable may be used to test for the successful completion of a RXVSAM statement regardless of how RXVSAM was invoked. This facilitates the use of a common error routine to handle all RXVSAM errors.

VSAMMSG

The special variable **VSAMMSG** will always contain a descriptive message following any invocation of RXVSAM. This variable is available whether RXVSAM was invoked via the host command environment (**ADDRESS RXVSAM**), as an external routine (**CALL "RXVSAM"**), or as a function (**RXVSAM (...)**). This variable contains descriptive text that can be used to further assist in determining the problem.

```
CALL "RXVSAM DELETE FILE(TEST02) RIDFLD(RCDKEY)"
IF VSAMCODE<>0 THEN DO           /* error code returned here    */
  SAY 'VSAM return code =' VSAMCODE   /* show it                 */
  SAY 'and description  =' VSAMMSG    /* detailed message        */
END
```

Figure 18: Special Variable VSAMMSG

In the above example, the VSAM return code and the VSAM return code description are displayed following an unsuccessful attempt to execute the statement.

VSMTRACE

When the special variable **VSMTRACE** is assigned the value `'TRACE'`, RXVSAM will write the resolved statement to SYSTSPRT. This special trace uses the REXX **SAY** function and may be used in any REXX environment. Tracing can be stopped by assigning **VSMTRACE** to any value other than `'TRACE'`. This tracing will occur whether RXVSAM is invoked via the host command environment (**ADDRESS RXVSAM**), an external routine (**CALL "RXVSAM"**), or as a function (**RXVSAM (...)**).

```
VSMTRACE='TRACE'              /* turn on the trace facility   */
```

Figure 19: Special Variable VSMTRACE

Open and Close Statements

Open a File (OPEN)

The **OPEN** command initializes the RXVSAM host command environment (if it is not already initialized by a previous **OPEN**) and opens a file (unless the `DUMMY` operand is specified). It also relates the file name with a copybook layout or mapping criteria for subsequent RXVSAM statements. Up to 100 files may be opened at any one time. **OPEN** may only be invoked via the **CALL "RXVSAM"** external function, or the **RXVSAM ( )** function call.

When opening a file that will contain transformed data, the LRECL of that file must be large enough to hold the largest input record. It is recommended that when transformed data is being written to an MVS data set, that the data set be defined as `RECFM=VB`, `BLKSIZE=32760`, `LRECL=32756`. When opening a file to write transformed data, the copybook or **MAPDD** to define the data must be specified.

When transformed data is being written to a UNIX file, a 'dummy' type **OPEN** must be issued for a unique file name in order to relate the output data to a map or copybook. The **FORMAT** and **CODEPAGE** are then specified on this dummy open. See the return operand on the **FORMAT** statement for details on the content of the transformed record.

Command Format

```
RXVSAM OPEN
        FILE(ddname)
        [ COPYBOOK('dsn(mem)')|PLICOPY('dsn(mem)')|MAPDD(mapdd)          ]
        [ INPUT|UPDATE|LOAD|SEQ|SEQUPD|DUMMY                            ]
        [ FORMAT(format)                                                ]
        [ CODEPAGE(cpage)                                               ]
```

Parameters

ddname	Specifies the ddname of the data set to be accessed. The name must be alphanumeric, up to 8 characters in length. The **ddname** must have been previously allocated unless keyword `DUMMY` is specified.
'dsn(mem)'	Specifies a pre-existing sequential file, or partitioned data set and member name, that contains the COBOL layout (`"COPYBOOK"`), or PL/I layout (`"PLICOPY"`), for the file. The data set must have a record length of 80. This data set name may be enclosed in quotes. Whether or not quotation marks are used, the `TSO USER-ID` prefix will not be used to prefix the data set name.
mapdd	Specifies the **ddname** of a mapping criteria data set and member name. The name must be alphanumeric, up to 8 characters in length. The **mapdd** must have been previously allocated. Mapping criteria allows multiple record layouts for a single file. The MAX Data/Util product is used to build a mapping criteria member.

INPUT	Specifies that the associated data set is to be opened for input processing only. When opened for input processing **DELETE**, **REWRITE**, and **WRITE** operations are not allowed. This is the default.
SEQ	Specifies that the associated data set is to be opened for input sequential forward processing only. Only the **READNEXT** operation is allowed. Use of this operand can substantially improve sequential performance.
SEQUPD	Specifies that the associated data set is to be opened for update in a sequential mode. This allows the file to be read sequentially forward, at the same time allowing records to be updated. Not all records need be updated; selection can be done to determine which records require update. Only the **READNEXT** and **REWRITE** operations will be allowed. Use of this operand can substantially improve sequential update performance.
UPDATE	Specifies that the associated data set is to be opened for update processing. When opened for update processing **DELETE**, **REWRITE**, and **WRITE** operations are allowed.
LOAD	Specifies the associated data set is opened in load mode. Load mode is used when a VSAM data set is empty and the initial record(s) need to be loaded. Only the **WRITE** operation is allowed. Load mode is also used to write to a sequential file.
DUMMY	Specifies that the file is not to be actually opened, but rather relates a record format (`COPYBOOK`) or multiple record formats (`MAPDD`) to a **ddname** (`FILE`) so that the record and field formatting commands (**GETFIELD**, **PUTFIELD**, **FORMAT**,**SELECTCB**) may be used.
format	Specifies the data transformation format of the output file. This can be XML, CSV or TAB, the templates that are shipped with the product. This can also be a user-defined template name.
cpage	Specifies the code page value to be used in data transformation.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE Return code from last operation.

`0`	Successful
`-3`	Syntax error
`12`	Unable to process statement

RC Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).

RESULT Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).

VSAMMSG Descriptive message.

Upon the successful completion, RXVSAM will set the following REXX variables to new values.

ddname_KEYLENGTH	Length of key field.
ddname_KEYPOS	Key position within record, relative to the first character.
ddname_TYPE	If DSORG is VS, this variable will indicate the type of VSAM file. Type: KSDS, ESDS, RRDS, SAM Example: K
ddname_RCDLENGTH	Maximum record length.
ddname_DSORG	Data set organization: PO, PS, DA, VS.
ddname_RECFM	Record format: **[Variable|Undefined] [Block] [Spanned|Ansi|Machine]** Examples: FBA, VB, FM, etc (up to three characters).
ddname_BLKSIZE	Block size of data set.
ddname_PATH	Indicates if data set is a VSAM path. **U** unique path **N** non unique path **null** not a path
ddname_DSNAME	Data set name.

The following variables are returned only when a file is opened with the **MAPDD** option.

ddname_CBNAME.0	Count of copybooks contained in mapping criteria member.
ddname_CBNAME.n	ddname of copybook entry.
ddname_CBDESC.n	Description of copybook entry.

```
                                /* allocate a VSAM data set       */
  ADDRESS TSO "ALLOCATE FILE(FILE1) DA('SAMPLE.VSAM.KSDS') SHR"
. . .
                                /* open file for updating         */
  CALL "RXVSAM" "OPEN FILE(FILE1) UPDATE"
. . .                           /*
                                /* update file from changed record*/
  ADDRESS RXVSAM "REWRITE FILE(FILE1) FROM (RCD)"   /*            */
                                /*                                */
. . .
  CALL "RXVSAM" "CLOSE FILE(FILE1)" /* close the file            */
. . .
  ADDRESS TSO "FREE FILE(FILE1)"   /* free the file allocation     */
```

Figure 20: Example Open for Update

```
FILEIN='SYSUT1'                        /* assign file ddname      */
MAPFILE='MAPDDNM'                      /*    and map ddname       */

CALL "RXVSAM" "OPEN FILE("FILEIN") MAPDD("MAPFILE")"

IF RC=0 THEN DO                        /* on successful OPEN      */
   SAY 'COPYBOOKS CONTAINED WITHIN MAP:'     /*                   */
   NDX=VALUE(FILEIN'_CBNAME.0')        /* number of copybooks     */
   DO I=1 TO NDX                       /* display all of the      */
      SAY VALUE(FIELIN'_CBNAME.'I)     /*   copybook ddnames      */
      SAY VALUE(FIELIN'_CBDESC.'I)     /*   and descriptions      */
   END                                 /*                         */
END                                    /*                         */
```

Figure 21: Example Open using MAPDD

```
RC=RXVSAM("OPEN FILE(DUMMAP) MAPDD(MAPDD) DUMMY FORMAT(XML)")
IF RC<>0 THEN DO
  SAY 'VSAMCODE='VSAMCODE
  SAY 'VSAMMSG='VSAMMSG
  CALL CLEANUP
END
```

Figure 22: Example of Open for a dummy file to be used to transform data to be returned for writing to UNIX

```
IF "RXVSAM"("OPEN FILE(SYSUT2) MAPDD(MAPDD)",
      " LOAD FORMAT(XML)")<>0 THEN DO
   SAY 'RC='VSAMCODE 'MSG='VSAMMSG
   CALL CLEANUP
END
```

Figure 23: Example Open of an MVS data set to receive transformed data

Close a File (CLOSE)

The **CLOSE** command terminates the RXVSAM host command environment (when no other files are open) and closes a previously opened file. **CLOSE** may only be invoked via the **CALL "RXVSAM"** external function, or the **RXVSAM**() function call.

Command Format

```
RXVSAM CLOSE
          FILE(ddname)
          [ RETURN(rtnarea)                    ]
```

Parameters

ddname	Specifies the **ddname** of the data set to be accessed. The name must match the file specified at **OPEN**.
rtnarea	Variable name to receive any final transformation data returned when the file is closed. This variable is valid only when the transformed data is being written to a UNIX file by the MAX/REXX program.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE Return code from last operation.

0	Successful
-3	Syntax error
12	Unable to process statement

RC Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).

RESULT Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).

VSAMMSG Descriptive message.

```
. . .
                                /* open file for updating        */
  CALL "RXVSAM" "OPEN FILE(FILE1) UPDATE"
. . .                           /*
                                /* update file from changed record*/
  ADDRESS RXVSAM "REWRITE FILE(FILE1) FROM (RCD)"   /*           */
                                /*                               */
. . .
  CALL "RXVSAM" "CLOSE FILE(FILE2)" /* close the file            */
```

Figure 24: Close File

Record Access and Positioning

The following list of SAM/VSAM file record access and positioning statements. These statements may be processed by RXVSAM either via host command, the `CALL "RXVSAM"` external routine, or the **RXVSAM()** function call.

Statement	Open Format						Statement Function
	INPUT	LOAD	SEQ	SEQUPD	UPDATE	DUMMY	
DELETE					*		Delete a record.
READ	*				*		Read a record specific record by RID.
READNEXT	*		*	*	*		Read forward to next record.
READPREV	*				*		Read backwards to previous record.
REWRITE				*	*		Update record.
STARTBR	*				*		Position to record.
ENDBR	*				*		End sequential processing mode.
WRITE		*			*		Write a new record.

Delete a Record (DELETE)

Delete one record from the file. The record must have been previously read for update. **RIDFLD** keyword identifies the complete key of the record to be deleted.

Command Format

```
RXVSAM DELETE
       FILE(ddname)
       RIDFLD(rid_varname)
```

Parameters

ddname	Specifies the **ddname** of the data set to be accessed. The name must match the **FILE** specified at **OPEN**.
rid_varname	Identifies the name of a REXX variable that contains the record identification field. The contents must be a complete key when the file is KSDS, an 8-byte binary relative byte address when the file is an ESDS or a relative record number when the file is either RRDS or SAM.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE	Return code from last operation. `0` Successful `-3` Syntax error `12` Unable to process statement
RC	Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).
RESULT	Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).
VSAMMSG	Descriptive message.

```
. . .
                                  /* open file update using copybook*/
  CALL "RXVSAM" "OPEN FILE(FILE1)
       COPYBOOK('SAMPLE.VSAM.COPYLIB(LAYOUT1)')
       UPDATE"
. . .                             /*
/* ***  delete all customer numbers of less than 10000              */
                                  /*
  DO UNTIL REACRC<>0              /* loop thru entire file         */
     READRC=RXVSAM("READNEXT FILE(FILE1)")
     IF READRC<>0 THEN ITERATE    /* force out of loop             */
     IF CUSTNO>=10000 THEN ITERATE /* read again                   */
                                  /* reread record for update      */
     RC=RXVSAM("READ FILE(FILE1) RIDFLD(CUSTONO) UPDATE")
     IF RC<>0 THEN DO             /* if error                      */
. . .                             /* do any error processing       */
     END                          /* endif error                   */
                                  /* delete the record             */
     RC=RXVSAM("DELETE FILE(FILE1) RIDFLD(CUSTNO)")
     IF RC<>0 THEN DO             /* if error                      */
. . .                             /* do any error processing       */
     END                          /* endif error                   */
  END                             /*                               */
                                  /*                               */
. . .
  CALL "RXVSAM" "CLOSE FILE(FILE1)" /* close the file              */
. . .
```

Figure 25: Delete a record

Read a Record (READ)

Read a record in direct mode. The **RIDFLD** keyword identifies the complete or partial key of the record to be read. If specified, the variable name identified in the **INTO** control option will be used to store the just read record; otherwise, variables as identified by a copybook layout, or a copybook selected from mapping criteria will be formatted with data from the record. This form of the **READ** is for use with SAM and VSAM files. To read forward, use **READNEXT**.

Command Format

```
RXVSAM READ
          FILE(ddname)
          RIDFLD(rid_varname)
          [ INTO(rcd_varname)                    ]
          [ GTEQ | EQUAL                         ]
          [ UPDATE                               ]
```

Parameters

ddname	Specifies the **ddname** of the data set to be accessed. The name must match the **FILE** specified at **OPEN**.
rid_varname	Identifies the name of a REXX variable that contains the record identification field. The contents must be a complete key when the file is KSDS, an 8 byte binary relative byte address when the file is an ESDS or a relative record number when the file is either RRDS or SAM.
rcd_varname	Identifies the name of a REXX variable into which the data is to be placed following the successful completion of the requested operation. When this keyword is used, the data contained in the layout (COPYBOOK or MAPDD) variables is not used.
GTEQ	Specifies that if the search for a record having the same key (complete or generic) as that specified in the **RIDFLD** keyword is unsuccessful, the first record having a greater key will satisfy the search. This is the default.
EQUAL	Specifies that the search will be satisfied only by a record having the same key (complete or generic) as that specified in the **RIDFLD** keyword
UPDATE	Specifies that the record is to be obtained for updating or deletion.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE Return code from last operation.

`0`	Successful
`-3`	Syntax error
`4`	File was opened with MAPDD, no copybook match found. Record returned in **ddname_INTO** variable.
`8`	End of file
`12`	Unable to process statement

RC Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).

RESULT Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).

VSAMMSG Descriptive message.

Upon the successful completion of this command, RXVSAM will set the following REXX variables to new values.

ddname_RCDLENGTH Record length current record.

ddname_RIDFLD Key of record. For the entry sequenced data set (ESDS) it contains the 8 byte binary relative byte address (RBA). For key sequenced data sets (KSDS), it will contain the data KEY. For relative record data sets (RRDS) and sequential files it will contain the relative record number (RRN).

The following variables are returned only if the file is opened with the MAPDD option:

ddname_CBDD **ddname** of copybook entry chosen.

ddname_DESC description of copybook chosen.

ddname_INTO (only with `RC=4`) no match on copybook, contains the entire record.

```
. . .
                                  /* open file for update          */
  CALL "RXVSAM" "OPEN FILE(FILE1)
       UPDATE"
. . .                             /*
  CUSTNO=10001                    /* identify record to read       */
                                  /*
  READRC=RXVSAM("READ FILE(FILE1) RIDFLD(CUSTNO)",
                "INTO(CUST_RCD) EQUAL"
. . .
  CALL "RXVSAM" "CLOSE FILE(FILE1)" /* close the file              */
. . .
```

Figure 26: Read a record

Read Next Record (READNEXT)

Read the next record in the file unless the previous RXVSAM command was not **READNEXT** or the **RIDFLD** operand specified has changed. In these cases, skip sequential forward processing will occur, the next record with a key greater than or equal is returned. If specified, the variable name identified in the **INTO** operand will be used to store the just read record; otherwise, variables as identified by the copybook layout or copybook chosen from the mapping criteria will be formatted with data from the record. Record selection criteria specified in the **WHERE** operand will allow only specific records to be returned.

Command Format

RXVSAM READNEXT
FILE(ddname)
[RIDFLD(rid_varname)]
[INTO(rcd_varname)]
[WHERE(where_clause)]
[WHERELIM(wherennn)]

Parameters

<table>
<tr><td>ddname</td><td colspan="2">Specifies the ddname of the data set to be accessed. The name must match the FILE specified at OPEN.</td></tr>
<tr><td>rid-varname</td><td colspan="2">Identifies the name of a REXX variable that contains the record identification field. The contents must be a complete key when the file is KSDS, an 8 byte binary relative byte address when the file is an ESDS or a relative record number when the file is either RRDS or SAM.</td></tr>
<tr><td>rcd_varname</td><td colspan="2">Identifies the name of a REXX variable into which the data is to be placed following the successful completion of the requested operation. When this keyword is used, data from the record is not formatted into copybook variable names.</td></tr>
<tr><td rowspan="3">where_clause</td><td colspan="2">The where_clause has the following formats:
(pos,cond,string)
(pos,cond,C'string,string')
(pos,cond,C'string',T'string')
(pos,cond,string,pos,cond,string)
(Pos,len,EQN|NEN|nEQP|nNEP)

Multiple formats can be ANDed or ORed together by using the connector of AND or OR.

Example: (pos,cond,string,AND|OR,pos,cond,c'string,string')</td></tr>
<tr><td>pos</td><td>Is the position within the record to begin the search for the string that follows. The position can be expressed in three formats:

nnn a numeric value, relative to 1, of the position in the record to search for the data
nnn-nnn numeric starting and ending positions within which to scan for the data.
nnn-0 numeric starting position with zero as the ending position allows for the scan to continue from the starting position to the end of the record.</td></tr>
<tr><td>cond</td><td>The condition to be tested may be expressed as any one of the following: ‘EQ’, ‘NE’, ‘LE’, ‘GE’, ‘GT’, ‘LT’, ‘NGT’, ‘NLT’.</td></tr>
</table>

where_clause *(continued)*	**string**	Data string to be compared to the data in the record. The data string can be coded in any one of the following formats: **'string'** string will be case sensitive **string** string will be case insensitive **X'hexstr'** pass a hexadecimal string of data **P'nnnnnn'** packed data, with a sign **U'nnnnnn'** packed data, unsigned **Z'nnnnnn'** zoned numeric data, with a sign **N'nnnnnn'** zoned numeric data, unsigned **H'nn'** halfword binary **F'nnnn'** fullword binary **C'charstring'** character string, case sensitive **T'textstring** character string (text), case insensitive :**varname** host variable
	len	Length of numeric or packed field to be tested in format five (`EQN`, `NEN`, `EQP`,`NEP`).
	EQN|NEN|nEQP|nNEP	Specify the data type to be searched for.
	EQN	Search for numeric data, length of **len** beginning at specified position.
	NEN	Search for nonnumeric data, length, of **len** beginning at the specified position.
	nEQP	Search for packed data at the specified position. If **len** has a value of zero, then a packed field of any valid length will be searched. If **len** has a value other than zero, the packed field must be that length. The '**n**' preceding the **EQP** is optional, defaulting to `1`. This is the number of continuous packed fields to be found in the search.
	nNEP	Search for the data that is not packed beginning at the indicted position, for the specified **len**. If **len** is `0`, packed data of any valid length is allowed. The '**n**' preceding the **NEP** indicates the number of contiguous fields to search for, this field is optional, and the default is `1`.

where_clause *(continued)*	Format 1: **(pos,cond,string)**	Is used to search for a single character string at the given position.
	Format 2: **(pos,cond,C'string, string')**	Is used to search for multiple character strings at a given location, this is an implied **OR**. For example, **(10,EQ,C'ABC,DEF')** will search for either an `ABC` or a `DEF` at location `10`, either true condition will be returned.
	Format 3: **(pos,cond,C'string', T'textstring')**	Is used to search for multiple strings of data at the same location using an implied **OR** (the same as format 2), but the data strings can be of different types.
	Format 4: **(pos,cond,string,pos, cond,string)**	Is used to search for an **OR** condition at multiple locations. An implied **OR** is used and is treated as one set of operands as are formats 2 and 3 above. Example (no implied **OR**): **(3,EQ,C'A1',AND,7,EQ,C'ABC',OR,19,EQ, C'CO')** This example would return records that contained `A1` in position `3` and `ABC` in position `7` or `CO` in position `19`. Example (implied **OR**): **(3,EQ,C'A1',AND,7,EQ,C'ABC',19,EQ,C'CO')** This example would return records that contained `A1` in position `3` in combination with `ABC` in position `7` or `CO` in position `19`.
	Format 5: **(pos,len,EQN,NEN, nEQP,nNEP)**	Is used to search for types of data strings, numeric or non-numeric, packed or not packed.
wherennn	A numeric value denoting the maximum number of records to search for the condition. If no record selection occurs within the limit specified, control is returned to the REXX program with a return code of `12`. The **ddname_COUNT** variable will contain the number of records that have been read.	

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE — Return code from last operation.

`0`	Successful
`-3`	Syntax error
`4`	File was opened with MAPDD, no copybook match found. Record returned in **ddname_INTO** variable.
`8`	End of file
`12`	Unable to process statement

RC — Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).

RESULT — Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).

VSAMMSG — Descriptive message.

Upon the successful completion of this command, RXVSAM will set the following REXX variables to new values.

ddname_RCDLENGTH — Record length current record.

ddname_RIDFLD — Key of record. For the entry sequenced data set (ESDS) it contains the 8 byte binary relative byte address (RBA). For key sequenced data sets (KSDS), it will contain the data key. For relative record data sets (RRDS) and sequential files it will contain the relative record number (RRN).

ddname_COUNT — A value of the number of records read. This can be useful when a **WHERE** clause is used to determine the actual number of records read for this specific command invocation. This count is reset for each **READNEXT**, and is not a cumulative value.

The following variables are returned only if the file is opened with the MAPDD option:

ddname_CBDD — **ddname** of copybook entry chosen.

ddname_DESC — Description of copybook chosen.

ddname_INTO (only with `RC=4`) — No match on copybook, contains the entire record.

```
. . .
                                  /* open file using copybook       */
  CALL "RXVSAM" "OPEN FILE(FILE1)
       COPYBOOK('SAMPLE.VSAM.COPYLIB(LAYOUT1)')"
. . .                             /*
  ADDRESS RXVSAM                  /* set default host environment   */
                                  /*
  CUSTNO=00101                    /* set first record to process    */
  "RXVSAM STARTBR FILE(FILE1) RIDFLD(CUSTNO)"
. . .
  DO UNTIL VSAMCODE<>0            /*                                */
     "RXVSAM READNEXT FILE(FILE1)"

. . .                             /* process the file               */

  END                             /*                                */
                                  /*                                */
. . .
  CALL "RXVSAM" "ENDBR"           /* end sequential mode processing */
. . .
  CALL "RXVSAM" "CLOSE FILE(FILE1)" /* close the file               */
```

Figure 27: **READNEXT** using **COPYBOOK**

```
. . .
                                  /* open file in sequential mode   */
  CALL "RXVSAM" "OPEN FILE(FILE1) SEQ)"
. . .                             /*
/* *** only retrieve the records that are type 1 or 2               */
/* *** NOTE record type is in position 10 of the record             */
                                  /*
. . .
  DO UNTIL VSAMCODE<>0            /* process thru the file          */
    RC=RXVSAM("READNEXT FILE(FILE1) INTO(RCD)",
       "WHERE(10,EQ,'1',OR,10,EQ,'2')")
    IF RC<>0 THEN DO              /*                                */
. . .                             /* process non zero return code   */
    END                           /*                                */
    RECORD_TYPE=SUBSTR(RCD,10,1)  /* extract record type            */
    IF RECORD_TYPE=1 THEN DO      /* process type 1 records         */
. . .                             /*                                */
    END                           /*                                */
    ELSE DO                       /* else process type 2 records    */
. . .                             /*                                */
    END                           /*                                */
  END                             /*                                */
. . .
  CALL "RXVSAM" "CLOSE FILE(FILE1)" /* close the file               */
. . .
```

Figure 28: **READNEXT** with **INTO**

Read Previous Record (READPREV)

Read the previous record in the file unless the previous RXVSAM command was not **READPREV** or the **RIDFLD** control option is specified. In these cases, skip sequential backward processing will occur until the next record with a key less than or equal to is returned. If specified, the variable name identified in the **INTO** control option will be used to store the just read record; otherwise, variables as identified by the copybook layout or copybook chosen from the mapping criteria will be formatted with data from the record.

Command Format

```
RXVSAM READPREV
          FILE(ddname)
          [RIDFLD(rid_varname)                    ]
          [ INTO(rcd_varname)                     ]
          [ WHERE(pos,cond,string)                ]
          [ WHERELIM(wherennn)                    ]
```

Parameters

ddname	Specifies the **ddname** of the data set to be accessed. The name must match the **FILE** specified at **OPEN**.	
rid-varname	Identifies the name of a REXX variable that contains the record identification field. The contents must be a complete key when the file is KSDS, an 8 byte binary relative byte address when the file is an ESDS or a relative record number when the file is either RRDS or SAM.	
rcd_varname	Identifies the name of a REXX variable into which the data is to be placed following the successful completion of the requested operation. When this keyword is used, data from the record is not formatted into copybook variable names.	
where_clause	The **where_clause** has the following formats: **(pos,cond,string)** **(pos,cond,C'string,string')** **(pos,cond,C'string',T'string')** **(pos,cond,string,pos,cond,string)** **(Pos,len,EQN|NEN|nEQP|nNEP)** Multiple formats can be **AND**ed or **OR**ed together by using the connector of **AND** or **OR**. Example: **(pos,cond,string,AND|OR,pos,cond,c'string,string')**	
	pos	Is the position within the record to begin the search for the string that follows. The position can be expressed in three formats: **nnn** a numeric value, relative to 1, of the position in the record to search for the data **nnn-nnn** numeric starting and ending positions within which to scan for the data. **nnn-0** numeric starting position with zero as the ending position allows for the scan to continue from the starting position to the end of the record.
	cond	The condition to be tested may be expressed as any one of the following: ‘EQ’, ‘NE’, ‘LE’, ‘GE’, ‘GT’, ‘LT’, ‘NGT’, ‘NLT’.

where_clause *(continued)*	**string**	Data string to be compared to the data in the record. The data string can be coded in any one of the following formats: **'string'** string will be case sensitive **string** string will be case insensitive **X'hexstr'** pass a hexadecimal string of data **P'nnnnnn'** packed data, with a sign **U'nnnnnn'** packed data, unsigned **Z'nnnnnn'** zoned numeric data, with a sign **N'nnnnnn'** zoned numeric data, unsigned **H'nn'** halfword binary **F'nnnn'** fullword binary **C'charstring'** character string, case sensitive **T'textstring** character string (text), case insensitive :**varname** host variable
	len	Length of numeric or packed field to be tested in format five (`EQN`, `NEN`, `EQP`,`NEP`).
	EQN|NEN|nEQP|nNEP	Specify the data type to be searched for.
	EQN	Search for numeric data, length of **len** beginning at specified position.
	NEN	Search for nonnumeric data, length, of **len** beginning at the specified position.
	nEQP	Search for packed data at the specified position. If **len** has a value of zero, then a packed field of any valid length will be searched. If **len** has a value other than zero, the packed field must be that length. The '**n**' preceding the **EQP** is optional, defaulting to `1`. This is the number of continuous packed fields to be found in the search.
	nNEP	Search for the data that is not packed beginning at the indicted position, for the specified **len**. If **len** is `0`, packed data of any valid length is allowed. The '**n**' preceding the **NEP** indicates the number of contiguous fields to search for, this field is optional, and the default is `1`.

where_clause *(continued)*	Format 1: **(pos,cond,string)**	Is used to search for a single character string at the given position.
	Format 2: **(pos,cond,C'string, string')**	Is used to search for multiple character strings at a given location, this is an implied **OR**. For example, **(10,EQ,C'ABC,DEF')** will search for either an `ABC` or a `DEF` at location `10`, either true condition will be returned.
	Format 3: **(pos,cond,C'string', T'textstring')**	Is used to search for multiple strings of data at the same location using an implied **OR** (the same as format 2), but the data strings can be of different types.
	Format 4: **(pos,cond,string,pos, cond,string)**	Is used to search for an **OR** condition at multiple locations. An implied **OR** is used and is treated as one set of operands as are formats 2 and 3 above. Example (no implied **OR**): **(3,EQ,C'A1',AND,7,EQ,C'ABC',OR,19,EQ, C'CO')** This example would return records that contained `A1` in position `3` and `ABC` in position `7` or `CO` in position `19`. Example (implied **OR**): **(3,EQ,C'A1',AND,7,EQ,C'ABC',19,EQ,C'CO')** This example would return records that contained `A1` in position `3` in combination with `ABC` in position `7` or `CO` in position `19`.
	Format 5: **(pos,len,EQN,NEN, nEQP,nNEP)**	Is used to search for types of data strings, numeric or non-numeric, packed or not packed.
wherennn	A numeric value denoting the maximum number of records to search for the condition. If no record selection occurs within the limit specified, control is returned to the REXX program with a return code of `12`. The **ddname_COUNT** variable will contain the number of records that have been read.	

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE Return code from last operation.

`0`	Successful
`-3`	Syntax error
`4`	File was opened with MAPDD, no copybook match found. Record returned in **ddname_INTO** variable.
`8`	End of file
`12`	Unable to process statement

RC Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).

RESULT Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).

VSAMMSG Descriptive message.

Upon the successful completion of this command, RXVSAM will set the following REXX variables to new values.

ddname_RCDLENGTH Record length current record.

ddname_RIDFLD Key of record. For the entry sequenced data set (ESDS) it contains the 8 byte binary relative byte address (RBA). For key sequenced data sets (KSDS), it will contain the data KEY. For relative record data sets (RRDS) and sequential files it will contain the relative record number (RRN).

ddname_COUNT A value of the number of records read. This can be useful when a **WHERE** clause is used to determine the actual number of records read for this specific command invocation.
This count is reset for each **READPREV**, and is not a cumulative value.

The following variables are returned only if the file is opened with the MAPDD option:

ddname_CBDD **ddname** of copybook entry chosen.

ddname_DESC Description of copybook chosen.

ddname_INTO (only with `RC=4`) No match on copybook, contains the entire record.

```
. . .
                                  /* open file using copybook       */
  CALL "RXVSAM" "OPEN FILE(FILE1)
       COPYBOOK('SAMPLE.VSAM.COPYLIB(LAYOUT1)')"
. . .                             /*
  ADDRESS RXVSAM                  /* set default host environment   */
                                  /*
  CUSTNO=99999                    /* set first record to process    */
  "RXVSAM STARTBR FILE(FILE1) RIDFLD(CUSTNO)"
. . .
  DO UNTIL VSAMCODE<>0            /* process thru the file          */
   "RXVSAM READPREV FILE(FILE1)"  /* in reverse direction     */

. . .                             /* process the file               */

  END                             /*                                */
                                  /*                                */
. . .
  CALL "RXVSAM" "ENDBR"           /* end sequential mode processing */
. . .
  CALL "RXVSAM" "CLOSE FILE(FILE1)" /* close the file               */
. . .
```

Figure 29: **READPREV** using a **COPYBOOK**

```
TESTFILE='SYSUT1'                          /* set the ddnames of the */
   MAPFILE='MAPDDN'                           /*   file and map         */
   RC=RXVSAM("OPEN FILE("TESTFILE") MAPDD("MAPFILE")")
 . . .                                        /*                        */
   FILEKEY='9999999'                          /* starting key value     */
   RC=RXVSAM("STARTBR FILE("TESTFILE") RIDFLD(FILEKEY)"   /* position */
 . . .                                        /*                        */
 /* *** read thru the file from starting position in reverse           */
                                              /*                        */
   DO UNTIL VSAMCODE>4                        /*                        */
     RC=RXVSAM("READPREV FILE("TESTFILE")")   /* read the record      */
     IF RC=4 THEN ITERATE                     /* not match, skip record */
     CBNAME=VALUE(TESTFILE'_CBDD')            /* name of copybook       */
     DESC=VALUE(TESTFILE'_DESC') /* description of copybook*/
     IF DESC='RECORD TYPE A' THEN DO          /* only look at a records */
         IF TYPE_A_CITY='NEW YORK' THEN       /* show all zipcodes  for */
            SAY 'ZIPCODE='TYPE_A_ZIP'         /*    for NEW YORK        */
     END                                      /*                        */
   END                                        /*                        */
                                              /*                        */
   IF VSAMCODE>8 THEN DO                      /* if not end of file     */
    SAY 'VSAMCODE='VSAMCODE /* show the error code    */
    SAY 'VSAMMSG=='VSAMMSG /*     and description*/
  END                                        /*                       */
                                             /*                       */
  RC=RXVSAM("CLOSE FILE(TESTFILE)")          /* close file when done  */
```

Figure 30: **READPREV** using a MAPDD

Update a Record (REWRITE)

Rewrite the last record read. If a VSAM or SAM file was opened "**UPDATE**," the previously issued **READ** must specify the **UPDATE** operand. When processing a VSAM or SAM file that was opened "**UPDATE**," a **READNEXT** may be followed by a **REWRITE** if the record is to be updated. If specified, the variable name identified in the **FROM** control option, will be used to replace the record in the file. If the file was opened using MAPDD, the copybook name identifying the record layout must be specified. This name was returned with the **READ** or **READNEXT** that returned the record. Otherwise, variables as identified by the copybook layout will be used to format a record that will be used to replace the record in the file.

Command Format

RXVSAM REWRITE
FILE(ddname)
[FROM(rcd_varname)]
[LENGTH(rcd_length)]
[CBDD(cbddname)]

Parameters

ddname	Specifies the **ddname** of the data set to be accessed. The name must match the **FILE** specified at **OPEN**.
rcd_varname	Identifies the name of a REXX variable from which the record is to be fetched. Overrides the record that is kept in-storage by RXVSAM. When this keyword is used, the data contained in the layout (COPYBOOK) variables is not used.
rcd_length	Specifies the length of the record to be written. It must be specified when writing variable length records.
cbddname	Specifies the **ddname** of the copybook associated with current record when file was opened with MAPDD option. This copybook name is returned from the read that returned the record.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE Return code from last operation.

0	Successful
-3	Syntax error
12	Unable to process statement

RC Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).

RESULT Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).

VSAMMSG Descriptive message.

```
. . .
                                  /* open file for update           */
  CALL "RXVSAM" "OPEN FILE(FILE1)  /* use a copybook for field names */
      COPYBOOK('SAMPLE.VSAM.COPYLIB(LAYOUT1)')
       UPDATE"
. . .                             /*
  CUSTNO=10001                    /* identify record to read        */
                                  /*
  READRC=RXVSAM("READ FILE(FILE1) RIDFLD(CUSTNO)",
                "EQUAL UPDATE"
. . .
  YTD_AMT=0                       /* reset the year-to-date amount  */

  CALL "RXVSAM" "REWRITE FILE(FILE1)"  /* update the file           */
. . .
  CALL "RXVSAM" "CLOSE FILE(FILE1)" /* close the file               */
. . .
```

Figure 31: Update file using **READ** and **REWRITE** and COPYBOOK

```
. . .
                                  /* open file for update          */
  CALL "RXVSAM" "OPEN FILE(FILE1)  UPDATE" /* not using copybook     */
. . .                             /*
  CUSTNO=10001                    /* identify record to read       */
                                  /*
  READRC=RXVSAM("READ FILE(FILE1) RIDFLD(CUSTNO)",
                "EQUAL UPDATE INTO(RCD)"
. . .                             /* notice that without a record  */
                                  /* layout, that the exact position*/
                                  /* of the data must be known     */
  RCD=OVERLAY(RCD,'000000',RCD,50,6)   /* reset the year-to-date amt */
 CALL "RXVSAM" "REWRITE FILE(FILE1) FROM(RCD)"  .
  CALL "RXVSAM" "CLOSE FILE(FILE1)" /* close the file                */
. . .
```

Figure 32: Update file using **READ INTO**, **REWRITE FROM**

Start Browse (STARTBR)

Position (no read) into the file in direct mode and prepare for sequential file processing. The variable name specified in the **RIDFLD** contains the key of the record to be located. A subsequent **READNEXT** or **READPREV** will begin from this record.

Command Format

RXVSAM STARTBR
FILE(ddname)
RIDFLD(rid_varname)
[GTEQ | EQUAL]

Parameters

ddname	Specifies the **ddname** of the data set to be accessed. The name must match the **FILE** specified at **OPEN**.
rid_varname	Identifies the name of a REXX variable that contains the record identification field. The contents must be a complete key when the file is KSDS, an 8 byte binary relative byte address when the file is an ESDS or a relative record number when the file is either RRDS or SAM.
GTEQ	Specifies that if the search for a record having the same key (complete or generic) as that specified in the **RIDFLD** keyword is unsuccessful, the first record having a greater key will satisfy the search. This is the default.
EQUAL	Specifies that the search will be satisfied only by a record having the same key (complete or generic) as that specified in the **RIDFLD** keyword.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE	Return code from last operation. `0` Successful `-3` Syntax error `8` End file `12` Unable to process statement, if key **EQUAL** specified a record of equal value was not found
RC	Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).
RESULT	Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).
VSAMMSG	Descriptive message.

```
. . .
                                /* open file using copybook       */
  CALL "RXVSAM" "OPEN FILE(FILE1)
       COPYBOOK('SAMPLE.VSAM.COPYLIB(LAYOUT1)')"
. . .                           /*
  ADDRESS RXVSAM                /* set default host environment   */
                                /*
  CUSTNO=00101                  /* set first record to process    */
  "RXVSAM STARTBR FILE(FILE1) RIDFLD(CUSTNO)"
. . .
  DO UNTIL VSAMCODE<>0          /* process thru the file          */
     "RXVSAM READNEXT FILE(FILE1)"

. . .                           /* process the file               */

  END                           /*                                */
                                /*                                */
. . .
  CALL "RXVSAM" "ENDBR"         /* end sequential mode processing */
. . .
  CALL "RXVSAM" "CLOSE FILE(FILE1)" /* close the file             */
. . .
```

Figure 33: Start Browse to Position File for Processing

End Browse (ENDBR)

Terminates a previous **STARTBR** and ends sequential mode processing.

Command Format

rxvsam endbr
FILE(ddname)

Parameters

ddname	Specifies the **ddname** of the data set to be accessed. The name must match the **FILE** specified at **OPEN**.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE Return code from last operation.

0	Successful
-3	Syntax error
12	Unable to process statement

RC Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).

RESULT Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).

VSAMMSG Descriptive message.

```
. . .
                                  /* open file using copybook       */
  CALL "RXVSAM" "OPEN FILE(FILE1)
       COPYBOOK('SAMPLE.VSAM.COPYLIB(LAYOUT1)')"
. . .                             /*
  ADDRESS RXVSAM                  /* set default host environment   */
                                  /*
  CUSTNO=00101                    /* set first record to process    */
  "RXVSAM STARTBR FILE(FILE1) RIDFLD(CUSTNO)"
. . .
  DO UNTIL VSAMCODE<>0            /* process thru the file          */
     "RXVSAM READNEXT FILE(FILE1)"

. . .                             /* process the file               */

  END                             /*                                */
                                  /*                                */
. . .
  CALL "RXVSAM" "ENDBR"           /* end sequential mode processing */
. . .
  CALL "RXVSAM" "CLOSE FILE(FILE1)" /* close the file               */
. . .
```

Figure 34: End Browse

Write a Record (WRITE)

Add a record to the file. The variable name specified in the **RIDFLD** contains the complete key of the record that is to be added. If specified, the variable name identified in the **FROM** control option will be used to add the record in the file. If the file was opened using MAPDD, the copybook name identifying the record layout must be specified. This name was returned from the **SELECTCB** once the record is built. Otherwise, variables as identified by the copybook layout will be used to format a record and it will be used to add the record into the file.

Command Format

```
RXVSAM WRITE
          FILE(ddname)
          RIDFLD(rid_varname)
          [ FROM(rcd_varname)                    ]
          [ LENGTH(rcd_length)                   ]
          [ CBDD(ddname)                         ]
```

Parameters

ddname	Specifies the **ddname** of the data set to be accessed. The name must match the **FILE** specified at **OPEN**.
rid_varname	Identifies the name of a REXX variable that contains the record identification field. The contents must be a complete key when the file is KSDS, an 8 byte binary relative byte address when the file is an ESDS or a relative record number when the file is either RRDS or SAM.
rcd_varname	Identifies the name of a REXX variable from which the record is to be fetched. When this keyword is used, the data contained in the layout (COPYBOOK) variables is not used.
rcd_length	Specifies the length of the record to be written when processing a variable record length file.
cbddname	Specifies the copybook name within the mapping criteria that identifies this record. This can be determined by passing a record to the **SELECTCB** command.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE	Return code from last operation. `0` Successful `-3` Syntax error `12` Unable to process statement
RC	Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).
RESULT	Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).
VSAMMSG	Descriptive message.

```
. . .
                                  /* open file for update (or load) */
  CALL "RXVSAM" "OPEN FILE(FILE1)UPDATE"
. . .                             /*
  ADDRESS RXVSAM                  /* set default host environment   */
                                  /*
  CUSTNO=00101                    /* set record number to add       */
. . .                             /* build the record in variable   */
. . .                             /*   of CUST_RECORD               */
  "RXVSAM WRITE FILE(FILE1) RIDFLD(CUSTNO) FROM(CUST_RECORD)"
                                  /*                                */
. . .
  CALL "RXVSAM" "CLOSE FILE(FILE1)" /* close the file               */
. . .
```

Figure 35: Write a Record

```
OPENRC=RXVSAM("OPEN FILE(SYSUT2) COPYBOOK("DSNC")",
          "LOAD FORMAT(TAB)")

 . . . . .

 DO WHILE "RXVSAM"("READNEXT FILE(SYSUT1) INTO(REC_IN)")=0

   . . . . .

   RC=RXVSAM("WRITE FILE(SYSUT2)  FROM(REC_IN)")

   . . . . .


 END
```

Figure 36: Write a transformed record to an MVS data set

PDS Processing Statements

The following statements are available when processing Partitioned Data Sets.

Statement	Open Format			State Function
	INPUT	LOAD	UPDATE	
DIR	*		*	Load PDS Directory entries into variable array.
ADDMEM		*		Add a new member directory entry.
REPLMEM		*	*	Replace a member directory entry.
DELMEM			*	Delete a member directory.
RENAME			*	Rename a member.
FIND	*		*	Position to begin of a member.
READNEXT	*		*	Read forward to next record.
REWRITE			*	Update record.
WRITE		*		Write a new record.

Retrieve the Directory Information (DIR)

Retrieve the entire directory of a PDS. It will be returned in an array of variables along with the total number of entries. In addition to the member name, all directory information will be returned along with the member names.

Command Format

```
RXVSAM DIR
          FILE(ddname)
          INTO(dir_varname)
```

Parameters

ddname	Specifies the **ddname** of the data set to be accessed. The name must match the **FILE** specified at **OPEN**.
dir_varname	Identifies the REXX variable name array prefix into which the directory entries will be placed. A variable will be created for each entry in the directory along with a `'0'` level entry for the total count of the variables in the array. A suggested method would be to specify this as a stem variable. For example, **INTO(DIR_VAR.)**, would result in a series of variables built as follows: **DIR_VAR.1, DIR_VAR.2**, with **DIR_VAR.0** containing the maximum number of variables that were returned.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE Return code from last operation.

`0`	Successful
`-3`	Syntax error
`4`	Directory contains no members
`12`	Unable to process statement

RC Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).

RESULT Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).

VSAMMSG Descriptive message.

```
. . .
                                /* open the pds for input         */
  CALL "RXVSAM" "OPEN FILE(PDFFILE) INPUT)"
. . .                           /* read directory into stem       */
                                /* variables                      */
  CALL "RXVSAM" "DIR FILE(PDSFILE) INTO(DIR_ENT.)"
                                /*                                */
  DO MEM#=1 TO DIR_ENT.0        /* process thru each member       */
     SAY "Member Name="SUBSTR(DIR_ENT.MEM#,1,8)  /* show names       */
  END                           /*                                */
. . .
  CALL "RXVSAM" "CLOSE FILE(PDSFILE)"    /* close file when done     */
. . .
```

Figure 37: Retrieve Directory Information

Add a Member (ADDMEM)

Add a member to a PDS. A new member can be written to a PDS if it is open for load. Once all of the records for the member have been written to the file, the **ADDMEM** is issued to update the directory.

Command Format

```
RXVSAM ADDMEM
           FILE(ddname)
           MEMBER("member_name")
           [ USRDATA(userdata_varname)                    ]
```

Parameters

ddname	Specifies the **ddname** of the data set to be accessed. The name must match the **FILE** specified at **OPEN**.
member_name	Specifies the 8 character name of the new member to be added to the PDS.
userdata_varname	Identifies the name of a REXX variable into which the user data for the member has been built. This is an optional field and will be stored in the directory for this member.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE Return code from last operation.

`0`	Successful
`-3`	Syntax error
`4`	Member already exists
`12`	Unable to process statement

RC Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).

RESULT Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).

VSAMMSG Descriptive message.

```
********************************* Top of Data ********************************
/* REXX ************************************************************ */
/* DOC: copy a pds member from one dataset to another                */
/* ************************************************************** */
                                  /* allocate the files             */
  "ALLOCATE FI(SYSUT1) DA("MXS.MXRXVNNN.EXECS") SHR"
  IF RC<>0 THEN EXIT 4            /* simple error exit              */
  "ALLOCATE FI(SYSUT2) DA("MXS.MYTEST.EXECS") SHR"
  IF RC<>0 THEN EXIT 4            /* simple error exit              */
                                  /* open each file                 */
  CALL "RXVSAM" "OPEN FILE(SYSUT1) INPUT)"
  IF VSAMCODE<>0 THEN CALL CLEANUP /* cleanup on errors             */
  CALL "RXVSAM" "OPEN FILE(SYSUT2) LOAD)"
  IF VSAMCODE<>0 THEN CALL CLEANUP /* cleanup on errors             */
                                  /* locate the member to copy      */
  CALL "RXVSAM" "FIND FILE(SYSUT1) MEMBER(RXPDS2)"
  IF VSAMCODE<>0 THEN CALL CLEANUP /* cleanup on errors             */
                                  /* read the member                */
  DO WHILE RXVSAM("READNEXT FILE(SYSUT1) INTO(RCD)")<>0             */
     SAY RCC                      /* display each line of member    */
                                  /* write it to the output pds     */
     CALL "RXVSAM" "WRITE FILE(SYSUT2) FROM(RCD)"
     IF VSAMCODE<>0 THEN CALL CLEANUP /* cleanup on errors          */
  END                             /*                                */
                                  /* add member to output directory */
  CALL "RXVSAM" "ADDMEM FILE(SYSUT2) MEMBER(MYMEM)"                 */
  CALL CLEANUP                    /* clean up and exit              */
                                  /*                                */
/* ************************************************************** */
/*  clean up and exit the program                                 */
/* ************************************************************** */
                                  /*                                */
  IF VSAMCODE<>0 THEN             /* if error, show it              */
    SAY 'RXVSAM failed, RC='VSAMCODE VSAMMSG   /* along with detail */
                                  /*                                */
  CALL "RXVSAM" "CLOSE FILE(SYSUT1)"    /* close files before exit */
  CALL "RXVSAM" "CLOSE FILE(SYSUT2)"    /*                         */
                                  /*                                */
  "FREE FI(SYSUT1)"               /* free files                     */
  "FREE FI(SYSUT2)"               /* free files                     */
                                  /*                                */
  EXIT 0                          /*                                */
```

Figure 38: Add a Member

Replace a Member (REPLMEM)

Replace a member in a PDS. A member can be written to a PDS if it is open for load. Once all of the records for the member have been written to the file, the **REPLMEM** is issued to update the directory. This command can be used to replace an existing member or to add a new member.

Command Format

```
RXVSAM REPLMEM
          FILE(ddname)
          MEMBER(member_name)
          [ USRDATA(userdata_varname)                ]
```

Parameters

ddname	Specifies the **ddname** of the data set to be accessed. The name must match the **FILE** specified at **OPEN**.
member_name	Specifies the 8 character name of the member to be replaced in the PDS.
userdata_varname	Identifies the name of a REXX variable into which the user data for the member has been built. This is an optional field and is applicable only if the PDS has user data that is maintained by ISPF.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE Return code from last operation.

```
0     Successful
-3    Syntax error
 8    Member did not exist but has been added
12    Unable to process statement
```

RC Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).

RESULT Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).

VSAMMSG Descriptive message.

```
/* REXX ************************************************************ */
/* DOC: copy a pds member from one dataset; replace in another pds   */
/* ************************************************************** */
                                  /* allocate the files              */
  "ALLOCATE FI(SYSUT1) DA("MXS.MXRXVNNN.EXECS") SHR"
  IF RC<>0 THEN EXIT 4            /* simple error exit               */
  "ALLOCATE FI(SYSUT2) DA("MXS.MYTEST.EXECS") SHR"
  IF RC<>0 THEN EXIT 4            /* simple error exit               */
                                  /* open each file                  */
  CALL "RXVSAM" "OPEN FILE(SYSUT1) INPUT)"
  IF VSAMCODE<>0 THEN CALL CLEANUP /* cleanup on errors              */
  CALL "RXVSAM" "OPEN FILE(SYSUT2) LOAD)"
  IF VSAMCODE<>0 THEN CALL CLEANUP /* cleanup on errors              */
                                  /* locate the member to copy       */
  CALL "RXVSAM" "FIND FILE(SYSUT1) MEMBER(RXPDS2)"
  IF VSAMCODE<>0 THEN CALL CLEANUP /* cleanup on errors              */
                                  /* read the member                 */
  DO WHILE RXVSAM("READNEXT FILE(SYSUT1) INTO(RCD)")<>0             */
     SAY RCC                      /* display each line of member     */
                                  /* write it to the output pds      */
     CALL "RXVSAM" "WRITE FILE(SYSUT2) FROM(RCD)"
     IF VSAMCODE<>0 THEN CALL CLEANUP /* cleanup on errors           */
  END                             /*                                 */
                                  /* replace member in output dir.   */
  CALL "RXVSAM" "REPLMEM FILE(SYSUT2) MEMBER(MYMEM)"                */
  CALL CLEANUP                    /* clean up and exit               */
                                  /*                                 */
/* ************************************************************** */
/*  clean up and exit the program                                    */
/* ************************************************************** */
                                  /*                                 */
  IF VSAMCODE<>0 THEN             /* if error, show it               */
    SAY 'RXVSAM failed, RC='VSAMCODE VSAMMSG   /* along with detail  */
                                  /*                                 */
  CALL "RXVSAM" "CLOSE FILE(SYSUT1)"    /* close files before exit   */
  CALL "RXVSAM" "CLOSE FILE(SYSUT2)"    /*                           */
                                  /*                                 */
  "FREE FI(SYSUT1)"               /* free files                      */
  "FREE FI(SYSUT2)"               /* free files                      */
                                  /*                                 */
  EXIT 0                          /*                                 */
```

Figure 39: Replace Member

Delete a Member (DELMEM)

Delete a member from a PDS. A member can be deleted from a PDS if it is open for update.

Command Format

```
RXVSAM DELMEM
       FILE(ddname)
       MEMBER(member_name)
```

Parameters

ddname	Specifies the **ddname** of the data set to be accessed. The name must match the **FILE** specified at **OPEN**.
member_name	Specifies the 8-character name of the member to be deleted from the PDS.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE Return code from last operation.

`0`	Successful
`-3`	Syntax error
`8`	Member not found
`12`	Unable to process statement

RC Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).

RESULT Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).

VSAMMSG Descriptive message.

```
. . .
                                  /* open the pds for update        */
  CALL "RXVSAM" "OPEN FILE(PDFFILE) UPDATE)"
. . .                             /* read directory into stem       */
                                  /* variables                      */
  CALL "RXVSAM" "DIR FILE(PDSFILE) INTO(DIR_ENT.)"
                                  /*                                */
/* *** delete any member names that start with the letters 'OLD'     */
                                  /*                                */
  DO MEM#=1 TO DIR_ENT.0          /* process thru each member       */
     MEM_NAME=SUBSTR(DIR_ENT.MEM#,1,8)  /* extract member name       */
     IF SUBSTR(MEM_NAME,1,3)-'OLD' THEN DO   /* if name OLD......    */
       RC=RXVSAM('DELMEM FILE(PDSFILE) MEMBER('MEM_NAME')')
       IF RC<>0 THEN DO           /* if error                       */
. . .                             /* process any error              */
       END                        /*                                */
     END                          /*                                */
  END                             /*                                */
. . .
  CALL "RXVSAM" "CLOSE FILE(PDSFILE)"    /* close file when done     */
. . .
```

Figure 40: Delete Member

Rename a Member (RENAME)

Rename a member of a PDS.

Command Format

RXVSAM RENAME
FILE(ddname)
MEMBER(member_name)
NEWMEM(new_member)

Parameters

ddname	Specifies the **ddname** of the data set to be accessed. The name must match the **FILE** specified at **OPEN**.
member_name	Specifies the 8-character name of the member to be renamed in the PDS.
new_member	Specifies the new 8-character name for the member.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE	Return code from last operation. 0 Successful -3 Syntax error 4 Directory already contains member name 8 Member name not found 12 Unable to process statement
RC	Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).
RESULT	Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).
VSAMMSG	Descriptive message.

```
. . .
                                  /* open the pds for update        */
  CALL "RXVSAM" "OPEN FILE(PDFFILE) UPDATE)"
. . .                             /* read directory into stem       */
                                  /* varaibles                      */
  CALL "RXVSAM" "DIR FILE(PDSFILE) INTO(DIR_ENT.)"
                                  /*                                */
/* *** change any members names from 'OLD....' to 'NEW....'         */
                                  /*                                */
  DO MEM#=1 TO DIR_ENT.0          /* process thru each member       */
     MEM_NAME=SUBSTR(DIR_ENT.MEM#,1,8)  /* extrace member name       */
     IF SUBSTR(MEM_NAME,1,3)-'OLD' THEN DO   /* if name OLD......    */
       NEW_NAME=OVERLAY('NEW',MEM_NAME,1,3)  /* change to NEW        */
       RC=RXVSAM('RENAME FILE(PDSFILE) MEMBER('MEM_NAME'),
                 'NEWMEM('NEW_NAME')')       /* and issue the rename */
       IF RC<>0 THEN DO           /* if error                       */
. . .                             /* process any error              */
       END                        /*                                */
     END                          /*                                */
  END                             /*                                */
. . .
  CALL "RXVSAM" "CLOSE FILE(PDSFILE)"    /* close file when done     */
. . .
```

Figure 41: Rename a Member

Locate a Member (FIND)

In order to process records within a member of a PDS, it is necessary to position to the beginning of the member with a **FIND** command. By specifying the name of the member that is to be processed, and issuing the **FIND** command, the program will then be able to process the records specific to the member.

Command Format

```
RXVSAM FIND
         FILE(ddname)
         MEMBER( "member_name" )
```

Parameters

ddname	Specifies the **ddname** of the data set to be accessed. The name must match the **FILE** specified at **OPEN**.
member_name	Specifies the 8-character name of the member that is to be processed.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE Return code from last operation.

`0`	Successful
`-3`	Syntax error
`8`	Member not found
`12`	Unable to process statement

RC Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).

RESULT Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).

VSAMMSG Descriptive message.

```
/* REXX ************************************************************ */
/* DOC: display a pds member                                         */
/* ***************************************************************** */
                                  /* allocate the file               */
  "ALLOCATE FI(SYSUT1) DA("MXS.MXRXVNNN.EXECS") SHR"
  IF RC<>0 THEN EXIT 4            /* simple error exit               */
                                  /* open the file                   */
  CALL "RXVSAM" "OPEN FILE(SYSUT1) INPUT)"
  IF VSAMCODE<>0 THEN CALL CLEANUP /* cleanup on errors              */
                                  /* locate the member               */
  CALL "RXVSAM" "FIND FILE(SYSUT1) MEMBER(RXPDS2)"
  IF VSAMCODE<>0 THEN CALL CLEANUP /* cleanup on errors              */
                                  /* read the member                 */
  DO WHILE RXVSAM("READNEXT FILE(SYSUT1) INTO(RCD)")<>0             */
     SAY RCC                      /* display each line of member     */
  END                             /*                                 */
                                  /*                                 */
  CALL CLEANUP                    /* clean up and exit               */
                                  /*                                 */
/* ***************************************************************** */
/*  clean up and exit the program                                    */
/* ***************************************************************** */
                                  /*                                 */
  IF VSAMCODE<>0 THEN             /* if error, show it               */
    SAY 'RXVSAM failed, RC='VSAMCODE VSAMMSG   /* along with detail  */
                                  /*                                 */
  CALL "RXVSAM" "CLOSE FILE(SYSUT1)"    /* close file before exit    */
  "FREE FI(SYSUT1)"               /* free file                       */
                                  /*                                 */
  EXIT 0                          /*                                 */
```

Figure 42: Locate a Member

Read the Next Record (READNEXT)

Read the next record in the member. If specified, the variable name identified in the **INTO** operand will be used to store the just read record; otherwise, variables as identified by the copybook layout will be formatted with data from the record. Record selection criteria specified in the **WHERE** operand can allow only specific records to be returned.

Command Format

```
RXVSAM READNEXT
          [ INTO(rcd_varname)                ]
          [ WHERE(where_string)              ]
          [ WHERELIM(wherennn)               ]
```

Parameters

ddname	Specifies the **ddname** of the data set to be accessed. The name must match the **FILE** specified at **OPEN**.	
rcd_varname	Identifies the name of a REXX variable into which the data is to be placed following the successful completion of the requested operation. When this keyword is used, the data contained in the layout (COPYBOOK) variables is not used.	
where_clause	The **where_clause** has the following formats: **(pos,cond,string)** **(pos,cond,C'string,string')** **(pos,cond,C'string',T'string')** **(pos,cond,string,pos,cond,string)** **(Pos,len,EQN|NEN|nEQP|nNEP)** Multiple formats can be **AND**ed or **OR**ed together by using the connector of **AND** or **OR**. Example: **(pos,cond,string,AND|OR,pos,cond,c'string,string')**	
	pos	Is the position within the record to begin the search for the string that follows. The position can be expressed in three formats: **nnn** a numeric value, relative to 1, of the position in the record to search for the data **nnn-nnn** numeric starting and ending positions within which to scan for the data. **nnn-0** numeric starting position with zero as the ending position allows for the scan to continue from the starting position to the end of the record.
	cond	The condition to be tested may be expressed as any one of the following: ‘EQ’, ‘NE’, ‘LE’, ‘GE’, ‘GT’, ‘LT’, ‘NGT’, ‘NLT’.

where_clause *(continued)*	**string**	Data string to be compared to the data in the record. The data string can be coded in any one of the following formats: **'string'** string will be case sensitive **string** string will be case insensitive **X'hexstr'** pass a hexadecimal string of data **P'nnnnnn'** packed data, with a sign **U'nnnnnn'** packed data, unsigned **Z'nnnnnn'** zoned numeric data, with a sign **N'nnnnnn'** zoned numeric data, unsigned **H'nn'** halfword binary **F'nnnn'** fullword binary **C'charstring'** character string, case sensitive **T'textstring** character string (text), case insensitive :**varname** host variable
	len	Length of numeric or packed field to be tested in format five (`EQN`, `NEN`, `EQP`,`NEP`).
	EQN|NEN|nEQP|nNEP	Specify the data type to be searched for.
	EQN	Search for numeric data, length of **len** beginning at specified position.
	NEN	Search for nonnumeric data, length, of **len** beginning at the specified position.
	nEQP	Search for packed data at the specified position. If **len** has a value of zero, then a packed field of any valid length will be searched. If **len** has a value other than zero, the packed field must be that length. The '**n**' preceding the **EQP** is optional, defaulting to `1`. This is the number of continuous packed fields to be found in the search.
	nNEP	Search for the data that is not packed beginning at the indicted position, for the specified **len**. If **len** is `0`, packed data of any valid length is allowed. The '**n**' preceding the **NEP** indicates the number of contiguous fields to search for, this field is optional, and the default is `1`.

where_clause *(continued)*	Format 1: **(pos,cond,string)**	Is used to search for a single character string at the given position.
	Format 2: **(pos,cond,C'string, string')**	Is used to search for multiple character strings at a given location, this is an implied **OR**. For example, **(10,EQ,C'ABC,DEF')** will search for either an `ABC` or a `DEF` at location `10`, either true condition will be returned.
	Format 3: **(pos,cond,C'string', T'textstring')**	Is used to search for multiple strings of data at the same location using an implied **OR** (the same as format 2), but the data strings can be of different types.
	Format 4: **(pos,cond,string,pos, cond,string)**	Is used to search for an **OR** condition at multiple locations. An implied **OR** is used and is treated as one set of operands as are formats 2 and 3 above. Example (no implied **OR**): **(3,EQ,C'A1',AND,7,EQ,C'ABC',OR,19,EQ, C'CO')** This example would return records that contained `A1` in position `3` and `ABC` in position `7` or `CO` in position `19`. Example (implied **OR**): **(3,EQ,C'A1',AND,7,EQ,C'ABC',19,EQ,C'CO')** This example would return records that contained `A1` in position `3` in combination with `ABC` in position `7` or `CO` in position `19`.
	Format 5: **(pos,len,EQN,NEN, nEQP,nNEP)**	Is used to search for types of data strings, numeric or non-numeric, packed or not packed.
wherennn	A numeric value denoting the maximum number of records to search for the condition. If no record selection occurs within the limit specified, control is returned to the REXX program with a return code of `12`. The **ddname_COUNT** variable will contain the number of records that have been read.	

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE	Return code from last operation. `0` Successful `-3` Syntax error `8` End of member `12` Unable to process statement
RC	Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).
RESULT	Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).
VSAMMSG	Descriptive message.

Upon the successful completion of this command, RXVSAM will set the following REXX variables to new values.

ddname_RCDLENGTH	Record length current record.
ddname_COUNT	A value of the number of records read. This can be useful when a **WHERE** clause is used to determine the actual number of records read for this specific command invocation. This count is reset for each **READNEXT**, and is not a cumulative value.

```
/* REXX ************************************************************ */
/* DOC: display a pds member                                        */
/* ************************************************************** */
                                  /* allocate the file              */
  "ALLOCATE FI(SYSUT1) DA("MXS.MXRXVNNN.EXECS") SHR"
  IF RC<>0 THEN EXIT 4            /* simple error exit              */
                                  /* open the file                  */
  CALL "RXVSAM" "OPEN FILE(SYSUT1) INPUT)"
  IF VSAMCODE<>0 THEN CALL CLEANUP /* cleanup on errors             */
                                  /* locate the member              */
  CALL "RXVSAM" "FIND FILE(SYSUT1) MEMBER(RXPDS2)"
  IF VSAMCODE<>0 THEN CALL CLEANUP /* cleanup on errors             */
                                  /* read the member                */
  DO WHILE RXVSAM("READNEXT FILE(SYSUT1) INTO(RCD)")<>0            */
     SAY RCC                      /* display each line of member    */
  END                             /*                                */
                                  /*                                */
  CALL CLEANUP                    /* clean up and exit              */
                                  /*                                */
/* ************************************************************** */
/*  clean up and exit the program                                   */
/* ************************************************************** */
                                  /*                                */
  IF VSAMCODE<>0 THEN             /* if error, show it              */
    SAY 'RXVSAM failed, RC='VSAMCODE VSAMMSG   /* along with detail */
                                  /*                                */
  CALL "RXVSAM" "CLOSE FILE(SYSUT1)"    /* close file before exit   */
  "FREE FI(SYSUT1)"               /* free file                      */
                                  /*                                */
  EXIT 0                          /*                                */
```

Figure 43: Read Next Record

Update a Record (REWRITE)

Rewrite the last record read. When processing a PDS file that was opened "**UPDATE**," a **READNEXT** may be followed by a **REWRITE** if the record is to be updated. If specified, the variable name identified in the **FROM** control option will be used to replace the record in the file. Otherwise, variables as identified by the copybook layout will be used to format a record and it will be used to replace the record in the file.

Command Format

```
RXVSAM REWRITE
          FILE(ddname)
          [ FROM(rcd_varname)                    ]
          [ LENGTH(rcd_length)                   ]
```

Parameters

ddname	Specifies the **ddname** of the data set to be accessed. The name must match the **FILE** specified at **OPEN**.
rcd_varname	Identifies the name of a REXX variable from which the record is to be fetched. Overrides the record that is kept in-storage by RXVSAM. When this keyword is used, the data contained in the layout (COPYBOOK) variables is not used.
rcd_length	Specifies the length of the record to be written. It must be specified when writing variable length records.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE Return code from last operation.

`0`	Successful
`-3`	Syntax error
`12`	Unable to process statement

RC Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).

RESULT Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).

VSAMMSG Descriptive message.

```
/* REXX ************************************************************ */
/* DOC: update a pds member                                         */
/* ************************************************************** */
                                  /* allocate the file              */
  "ALLOCATE FI(SYSUT1) DA("MXS.MXRXVNNN.EXECS") SHR"
  IF RC<>0 THEN EXIT 4            /* simple error exit              */
                                  /* open file for update           */
  CALL "RXVSAM" "OPEN FILE(SYSUT1) UPDATE)"
  IF VSAMCODE<>0 THEN CALL CLEANUP /* cleanup on errors             */
                                  /* locate the member to update    */
  CALL "RXVSAM" "FIND FILE(SYSUT1) MEMBER(RXPDS2)"
  IF VSAMCODE<>0 THEN CALL CLEANUP /* cleanup on errors             */
                                  /* read the member                */
  DO WHILE RXVSAM("READNEXT FILE(SYSUT1) INTO(RCD)")<>0            */
                                  /* only changing 'last updated'   */
     IF SUBSTR(RCD,1,16)<>'/* Last updated:' THEN ITERATE
     RCD=OVERLAY(DATE(),RCD,18)   /* set with DATE function results */
     SAY RCD                      /* display results                */
     CALL "RXVSAM" "REWRITE FILE(SYSUT1) FROM(RCD)"
     IF VSAMCODE<>0 THEN CALL CLEANUP /* cleanup on errors          */
  END                             /*                                */
  CALL CLEANUP                    /* clean up and exit              */
                                  /*                                */
/* ************************************************************** */
/*  clean up and exit the program                                   */
/* ************************************************************** */
                                  /*                                */
  IF VSAMCODE<>0 THEN             /* if error, show it              */
    SAY 'RXVSAM failed, RC='VSAMCODE VSAMMSG   /* along with detail */
                                  /*                                */
  CALL "RXVSAM" "CLOSE FILE(SYSUT1)"    /* close file before exit   */
                                  /*                                */
  "FREE FI(SYSUT1)"               /* free files                     */
                                  /*                                */
  EXIT 0                          /*                                */
```

Figure 44: Update a Record

Field Access and Record Formatting

The following list of field access and record formatting statements may be processed by RXVSAM either via host command environment or the **CALL "RXVSAM"** external routine, or the **RXVSAM()** function call.

Statement	Open Format			Statement Function
	MAPDD	COPYBOOK	PLICOPY	
FORMAT FROM	*	*	*	Format new field variables from record.
FORMAT INTO	*	*	*	Format fields variables into new record.
GETFIELD	*	*	*	Fetch field from record.
PUTFIELD	*	*	*	Put field into record.
SELECTCB	*			Select matching copybook from **MAPDD**.

INTO and FROM Parameters Control Automatic Field Formatting

If **INTO** or **FROM** keyword parameters are not specified on a record access statement, then all fields (defined by a copybook, or a copybook component of mapping criteria) will be automatically formatted to/from REXX variables. When the **INTO** or **FROM** keyword parameters are specified automatic field formatting is bypassed and the entire record is stored-into/fetched-from a single REXX variable.

The **INTO** keyword may be specified on **READ** statements and the **FROM** keyword may be specified on **WRITE** statements. These features give the programmer complete control over automatic field formatting and can be used to maximize the performance of a program.

```
ADDRESS RXVSAM                    /* set default environment        */
                                  /* read record into a variable    */
"RXVSAM" "READNEXT FILE(TEST01) INTO(EMPLOYEE_RECORD)"
                                  /* fetch sepecific field from     */
                                  /* the variable containing record */
"RXVSAM" "GETFIELD FILE(TEST01) FIELD(EMPLOYEE_NAME)
         FROM(EMPLOYEE_RECORD)"
```

Figure 45: **INTO** and **FROM**

Format Variables from Record (FORMAT FROM)

Format all fields from the record into variables. If the file was opened with **MAPDD**, the copybook to use to format the record will be selected from the mapping criteria. If the file was opened with a copybook, the record will be formatted into the variables for that copybook.

Command Format

```
RXVSAM FORMAT
           FILE(ddname)
           FROM(rcd_varname)
           [ RETURN(rtnarea)                    ]
```

Parameters

ddname	Specifies the **ddname** of the data set to be accessed. The name must match the **FILE** specified at **OPEN**.
rcd_varname	Identifies the name of a REXX variable that contains the record to be formatted.
rtnarea	Specifies the variable name to receive the transformed data record. The **FILE** name used here is the copybook name that is associated with the record to be transformed.

Writing transformed data to a UNIX file.

When data is being transformed and returned to the MAX/REXX program to be written to a UNIX file, it is returned in the variable name specified in the **RETURN** operand. It can then be concatenated to any prior returned data until the buffer contains a multiple of 4K to be written to UNIX.

Note: The final data will be returned at the **CLOSE** that should be concatenated to any data remaining in the buffer and written to UNIX at that time.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE Return code from last operation.

`0`	Successful
`-3`	Syntax error
`4`	No copybook match found (only returned if file opened with **MAPDD** option)
`8`	End of file
`12`	Unable to process statement

RC Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).

RESULT Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).

VSAMMSG Descriptive message.

If the file was opened using the **MAPDD**, and a copybook is selected (`RC=0`) the following special variables are returned:

ddname_CBDD **ddname** of selected copybook.

ddname_DESC Description of selected copybook.

```
                                /* read record into a variable    */
"RXVSAM" "READNEXT FILE(TEST01) INTO(CUST_RECORD)"
                                /* format variables from that     */
                                /* record                         */
"RXVSAM" "FORMAT FILE(TEST01) FROM(CUST_RECORD)"
```

Figure 46: **FORMAT FROM**

```
TESTFILE='SYSUT1'                          /* set the ddnames of the */
  MAPFILE='MAPDDN'                           /*   file and map         */
  RC=RXVSAM("OPEN FILE("TESTFILE") MAPDD("MAPFILE")")
                                             /*                        */
/* *** read thru the file - FORMAT only the type A records            */
                                             /*                        */
  DO UNTIL VSAMCODE>0                        /*                        */
    RC=RXVSAM("READNEXT FILE("TESTFILE") INTO(RCD)"  /*                */
    IF SUBSTR(RCD,20,1)='A' THEN DO          /* only format type A recs*/
       RC=RXVSAM("FORMAT FILE("TESTFILE") INTO(RCD)"
       IF TYPE_A_CITY='NEW YORK' THEN      /* show all zipcodes  for */
           SAY 'ZIPCODE='TYPE_A_ZIP'         /*    for NEW YORK        */
    END                                      /*                        */
  END                                        /*                        */
                                             /*                        */
  IF VSAMCODE>8 THEN DO                      /* if not end of file     */
    SAY 'VSAMCODE='VSAMCODE /* show the error code    */
    SAY 'VSAMMSG=='VSAMMSG /*     and description*/
  END                                        /*                        */
                                             /*                        */
  RC=RXVSAM("CLOSE FILE(TESTFILE)")          /* close file when done   */
```

Figure 47: **FORMAT FROM** for file with **MAPDD** option

```
   RC=RXVSAM("SELECTCB FILE(DUMMAP) FROM(REC_IN)")
   IF RC<>0 THEN DO
       SAY 'VSAMCODE='VSAMCODE
       SAY 'VSAMMSG='VSAMMSG
       CALL CLEANUP
   END
                                          /* save copybook name     */
   CBNAME=VALUE(DUMMAP'_CBDD')
                                          /* pass rec to format     */
   RC=RXVSAM("FORMAT FILE("CBNAME") FROM(REC_IN) RETURN(XMLVAR)")
   IF RC<>0 THEN DO
       SAY 'RC='VSAMCODE 'MSG='VSAMMSG
       CALL CLEANUP
   END
```

Figure 48: **FORMAT** data for transformation when output is to be returned to write to UNIX and data has been defined using a map

Format Variables from Record (FORMAT INTO)

Reformat the record from all the REXX variables as defined in the copybook layout. The reformatted record will be placed in the variable identified by the **INTO** control option. If the **FILE** was opened with **MAPDD**, a specific copybook name must be passed in the **CBDD** operand.

Command Format

```
RXVSAM FORMAT
          FILE(ddname)
          INTO(rcd_varname)
          [CBDD(cbddname)                    ]
```

Parameters

ddname	Specifies the **ddname** of the data set to be accessed. The name must match the **FILE** specified at **OPEN**.
rcd_varname	Identifies the name of a REXX variable into which the record is to be formatted. Overrides the record that is kept in storage by RXVSAM.
cbddname	Specifies the name of a specific copybook within the mapping criteria that decribes this record. This is required if the file was opened with the **MAPDD** option. This name can be determined by passing the record to the **SELECTCB** command.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE Return code from last operation.
- `0` Successful
- `-3` Syntax error
- `12` Unable to process statement

RC Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).

RESULT Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).

VSAMMSG Descriptive message.

```
                              /* format variables into a record */
"RXVSAM" "FORMAT FILE(TEST01) INTO(CUST_RECORD_NEW)"
                              /* write a record                 */
"RXVSAM" "WRITE FILE(TEST02) RIDFLD(RCDKEY)",
         "FROM(CUST_RECORD_NEW)"
```

Figure 49: **FORMAT INTO**

Fetch a Specific Field from Record (GETFIELD)

Fetch and format a variable from a specific field in the record. The '`field_name`' must be a field that is defined in the copybook layout. If specified, the variable name identified in the **FROM** control option will be used as the record to fetch the field from; otherwise, the field will be fetched from the last read record. If the file was opened using **MAPDD**, the **GETFIELD** must specify the copybook **DDNAME** returned from a **SELECTCB** command in the **FILE** operand.

Command Format

```
RXVSAM GETFIELD
          FILE(ddname)
          FIELD(field_name)
          [ FROM(rcd_varname)                    ]
```

Parameters

ddname	Specifies the ddname of the data set to be accessed. The name must match the **FILE** specified at **OPEN**, or if opened with **MAPDD**, the name of the copybook **ddname** that matches the record.
field_name	Identifies by name the field from the layout (COPYBOOK) to be used for this operation.
rcd_varname	Identifies the name of a REXX variable from which the field is be fetched. Overrides the record that is kept in storage by RXVSAM.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE Return code from last operation.

`0`	Successful
`-3`	Syntax error
`12`	Unable to process statement

RC Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).

RESULT Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).

VSAMMSG Descriptive message.

```
                                /* read a record into a variable  */
"RXVSAM" "READNEXT FILE(TEST01) INTO(CUST_RECORD)"
                                /* fetch a specific field         */
"RXVSAM" "GETFIELD FILE(TEST01) FIELD(CUST_ID) FROM(CUST_RECORD)"
```

Figure 50: **GETFIELD**

Place a Specific Field in Record (PUTFIELD)

Place a specific field into the record. The '`field_name`' must be a field as defined in the copybook layout and must currently exist as a variable with the same name. If specified, the variable name identified in the **INTO** control option will be used as the record into which the field is to be placed; otherwise, the field will be placed into the last read record. If the file was opened using **MAPDD**, the ddname supplied must be that of the copybook describing this record.

Command Format

```
RXVSAM PUTFIELD
          FILE(ddname)
          FIELD(field_name)
          [ INTO(rcd_varname)                    ]
```

Parameters

ddname	Specifies the **ddname** of the data set to be accessed. The name must match the **FILE** specified at **OPEN**, or, if opened with **MAPDD**, the name of the copybook **ddname** that matches this record.
field_name	Identifies by name the field from the layout (COPYBOOK) to be used for this operation.
rcd_varname	Identifies the name of a REXX variable from which the field is be fetched. Overrides the record that is kept in storage by RXVSAM. This is required if the file was opened with the MAPDD option.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE Return code from last operation.

`0`	Successful
`-3`	Syntax error
`12`	Unable to process statement

RC Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).

RESULT Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).

VSAMMSG Descriptive message.

```
                                /* read a record into a variable  */
                                /* indicate for update            */
"RXVSAM" "READNEXT FILE(TEST01) INTO(CUST_RECORD) UPDATE"
                                /* fetch a specific field - AMT   */
"RXVSAM" "GETFIELD FILE(TEST01) FIELD(AMT) FROM(CUST_RECORD)"
                                /*                                */
AMT+AMT*1.15                    /* add 15% to AMT                 */
                                /* put value back into the record */
"RXVSAM" "PUTFIELD FILE(TEST01) FIELD(AMT) INTO(CUST_RECORD)"
                                /* rewrite the record             */
"RXVSAM" "REWRITE FILE(TEST01) FROM(CUST_RECORD)"
```

Figure 51: **PUTFIELD**

Select the Matching Copybook (SELECTCB)

Select the copybook that matches a specific record. When a file is opened with the **MAPDD**, multiple copybooks are associated with the file. When reading the file, if no **INTO** operand is specified, the records will be returned in the matching record format. When a record is read with the **INTO** operand or the record is built for output, it is necessary to pass the record through the selection process to obtain the ddname of the copybook that describes the record.

Command Format

RXVSAM SELECTCB
FILE(ddname)
FROM(rcd_varname)

Parameters

ddname	Specifies the **ddname** of the **FILE** that was opened with the **MAPDD** option.
rcd_varname	Identifies the name of a REXX variable that contains the record.

Special Variables

Upon completion, RXVSAM will set the following REXX variables to new values.

VSAMCODE	Return code from last operation. 0 Successful 4 No copybook match found 12 Unable to process statement
RC	Same as **VSAMCODE** when invoked as command (**ADDRESS RXVSAM**).
RESULT	Same as **VSAMCODE** when invoked as external subroutine (**CALL "RXVSAM"**).
VSAMMSG	Descriptive message.
ddname_CBDD	Contains **ddname** of selected copybook when successful.
ddname_DESC	Description of the selected copybook.

```
/*                        */
/* *** OPEN the map as a DUMMY file, OPEN the input files without    */
/* *** a COPYBOOK or MAP associated with them.  Use SELECTCB when    */
/* *** decision is made to format a record to select the copybook.   */
/* *** NOTE: always do approriate error processing after RXVSAM      */
/* ***       access.  Example does not show error checking.          */
                                        /*                          */
  RC=RXVSAM("OPEN FILE(SYSUT1) INPUT")     /* open the two           */
  RC=RXVSAM("OPEN FILE(SYSUT2) INPUT")     /*      input files       */
  RC=RXVSAM("OPEN FILE(MAP) MAPDD(MAPDDN) DUMMY)") /* and one map    */
                                        /*                          */
  RC=RXVSAM("READNEXT FILE(SYSUT1) INTO(RCD)")  /* read the record   */
                                        /*                          */
/* *** do processing that determines you want to map the record      */
/* *** pass the record to the SELECTCB to find the copybook          */
                                        /*                          */
  RC=RXVSAM("SELECTCB FILE(MAP) FROM(RCD)") /* select the copybook  */
  CBNAME=VALUE(DD'_CBDD') /* ddname of match        */
                                        /*                          */
  IF RXVSAM("FORMAT FILE("CBNAME") FROM(RCD)")<>0 THEN DO
     SAY 'VSAMCODE='VSAMCODE            /*                          */
     SAY 'VSAMMSG='VSAMMSG              /*                          */
  END                                   /*                          */
                                        /*                          */
/* *** record returned in corresponding copybook variables           */
                                        /*                          */
```

Figure 52: Use **SELECTCB** to find matching copybook

Requesting ISPF Services

You do not have to switch host command environments to invoke ISPF services. Precede any ISPF service with **ISPEXEC** and RXVSAM will invoke the ISPF service in its own host command environment.

The return codes are the same as if the ISPF service was invoked without RXVSAM.

```
ADDRESS RXVSAM                          /* set default environment  */
                                        /* display ISPF panel       */
"ISPEXEC DISPLAY PANEL(DENTRY)"
IF RC<>0 THEN ....
```

Figure 53: Requesting ISPF Services

Requesting PDF Edit Services

You do not have to switch host command environments to invoke PDF Edit services. Precede any PDF Edit service with **ISREDIT** and RXVSAM will invoke the PDF Edit service in its own host command environment.

The return codes are the same as if the PDF Edit service was invoked without RXVSAM.

```
ADDRESS RXVSAM                          /* set default environment  */
                                        /* begin Edit Macro         */
"ISREDIT MACRO (OPT) NOPROCESS"
IF RC<>0 THEN ....
```

Figure 54: Requesting PDF Edit Services

CHAPTER 4: SQL PROCESSING

Summary

Refer to the DB2 SQL reference manual for the complete SQL syntax. The following SQL statements are supported by RXSQL.

DB2 Connection Statements
CONNECT
DISCONNECT

SQL Administration Statements
ALTER
COMMENT
CREATE
DROP
GRANT
LABEL
LOCK
REVOKE

SQL Data Update Statements	
DELETE	(Searched **DELETE**)
DELETE	(Positioned **DELETE**) **WHERE CURRENT OF**
UPDATE	(Searched **UPDATE**)
UPDATE	(Positioned **UPDATE**) **WHERE CURRENT OF**
INSERT	

SQL Query Statements
DECLARE CURSOR
OPEN CURSOR
FETCH CURSOR
CLOSE CURSOR
SELECT INTO :host variables
SELECT INTO ISPF table
SELECT INTO STEM variable array
SELECT INTO FILE

SQL Recovery Statements
COMMIT
ROLLBACK

SQL Miscellaneous Statements
DECLARE STATEMENT
EXPLAIN
EXECUTE IMMEDIATE
EXECUTE USING :host variables
EXECUTE USING FILE
PREPARE
SET CURRENT
DESCRIBE

Special Variables

SQLCODE

The special variable **SQLCODE** will always contain the return code following any invocation of RXSQL. This variable is available whether RXSQL was invoked via the host command environment (**ADDRESS RXSQL**), as an external routine (**CALL "RXSQL"**), or as a function (**RXSQL (...)**). This variable may be used to test for the successful completion of a RXSQL statement regardless of how RXSQL was invoked. This facilitates the use of a common error routine to handle all RXSQL errors.

```
RC=RXSQL("GRANT TRACE TO PUBLIC")
IF SQLCODE<>0 THEN DO
   SAY 'SQL RETURN CODE='SQLCODE
   SAY 'RC DESCRIPTION ='SQLMSG
END
```

Figure 55: SQL Return Code

In the above example, the SQL return code and the SQL return code description are displayed following an unsuccessful attempt to execute the SQL statement.

SQLMSG

The special variable **SQLMSG** will always contain a descriptive message following any invocation of RXSQL. This variable is available whether RXSQL was invoked via the host command environment (**ADDRESS RXSQL**), an external routine (**CALL "RXSQL"**), or as a function (**RXSQL (...)**). This variable contains descriptive text that can be used to further assist in determining the problem.

```
CALL "RXSQL" "GRANT TRACE TO PUBLIC"
IF SQLCODE<>0 THEN DO
   SAY 'SQL RETURN CODE='SQLCODE
   SAY 'RC DESCRIPTION ='SQLMSG
END
```

Figure 56: SQL Return Code

In the above example, the SQL return code and the SQL return code description are displayed following an unsuccessful attempt to execute a statement.

SQLERRD.

The special variables **SQLERRD.1**, **SQLERRD.2**, **SQLERRD.3**, **SQLERRD.5** and **SQLERRD.6** will always contain the integer values of the **SQLERRD** fields of the DB2 SQLCA following any invocation of RXSQL. The entire **SQLERRD** stem is initialized to 0 at the start of each invocation of RXSQL.

SQLSTATE

The special variable **SQLSTATE** will always contain the five-character value of the DB2 **SQLSTATE** register following any invocation of RXSQL.

SQLTRACE

When the special variable **SQLTRACE** is assigned the value `TRACE`, RXSQL will write the resolved statement to **SYSTSPRT**. This special trace uses the REXX **SAY** function and may be used in any REXX environment. Tracing may be stopped by assigning **SQLTRACE** to any value other than `TRACE`. This tracing will occur whether RXSQL is invoked via the host command environment (**ADDRESS RXSQL**), an external routine (**CALL "RXSQL"**), or as a function (**RXSQL (...)**).

```
SQLTRACE="TRACE"
```

Figure 57: SQL Trace

SQLWARN

The special variable **SQLWARN** will always contain the eleven-character value of the **SQLWARN** field of the DB2 SQLCA following any invocation of RXSQL.

Host Variables

Referencing Host Variables

RXSQL statements may include a reference to a host variable. A host variable directs RXSQL to resolve the specified REXX variable name(s) that were passed as part of the command string. To use host variables, a ‘:’ (colon) must be placed immediately in front of the host variable name.

```
"RXSQL INSERT INTO CUSTOMER
       VALUES (:FNAME, :LNAME, :MI, :CITY, :STATE)"
```

Figure 58: Host Variables as parameters

Host variables are resolved in either of two ways:

1. As input, variable substitution will occur before execution.
2. As output, a variable will be assigned a new value as a result of execution.

Using Host Variables as Input

RXSQL resolves input variables as follows:

- If the variable value contains only displayable characters and numbers then it is resolved into a character string surrounded by apostrophes (`'...'`).
- If the variable value contains all numbers then it is resolved into a number.

```
LASTNAME='DOE'
FIRSTNAME='JOHN'
EMPNO='011'
"RXSQL UPDATE TEMP
    SET LASTNAME= :LASTNAME  FIRSTNAME= :FIRSTNAME
    WHERE EMPNO = :EMPNO"
```

Figure 59: Host Variables as Input

RXSQL will resolve the command before execution by DB2 to:

```
"RXSQL UPDATE TEMP SET LASTNAME = 'DOE'  FIRSTNAME = 'JOHN'
 WHERE EMPNO = 011"
```

If the resolution of input host variables performed by RXSQL is unsatisfactory for a specific SQL statement, then it may be necessary to dynamically build the SQL statement.

```
LASTNAME='DOE'
FIRSTNAME='JOHN'
EMPNO='011'
"RXSQL UPDATE TEMP",
   "SET LASTNAME= :LASTNAME",
       "FIRSTNAME= :FIRSTNAME",
 "WHERE EMPNO = '"EMPNO"'"
```

Figure 60: Host Variables as Input

RXSQL will resolve the command before execution by DB2 to:

```
"RXSQL UPDATE TEMP SET LASTNAME = 'DOE'  FIRSTNAME = 'JOHN'
WHERE EMPNO = '011'"
```

Figure 61: Statement after Variable Resolution

Using Host Variables As Output

RXSQL uses the **INTO** clause to determine when an output host variable has been specified.

```
"RXSQL SELECT MAX(SALARY) INTO :MAXSALRY FROM DSN8220.EMP"'
```

Figure 62: Host Variables as Output

After execution of the RXSQL statement, the variable **MAXSALRY** would be assigned a new value as a result of the query.

Parameter Markers

Parameter markers are determined when a '?' is encountered in a **DECLARE CURSOR** or **PREPARE** statement. They are resolved at **OPEN CURSOR** or **EXECUTE** time by specifying the **USING** clause with as many input host variables as '?' parameter markers.

```
"RXSQL DECLARE C1 CURSOR FOR
      SELECT DEPTNO, DEPTNAME, MGRNO, FROM DSN850.DEPT
      WHERE ADMRDEPT = ?"

DEPT='A11'
"RXSQL OPEN C1 USING :DEPT"
DO WHILE RXSQL("FETCH C1")
 . . .
 (print results)
 . . .
END

"RXSQL CLOSE C1"
```

Figure 63: Parameter Markers

Pulling It All Together

The following sample shows a program that uses input host variable, output host variables, parameter markers, multiple Cursors usage, along with full error handling.

```
/* REXX  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* DOC: RXSSAMP1 - SAMPLE RXSQL PROGRAM                                    */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

   CALL ON ERROR NAME ERRPROC              /* Trap command errors    */
   SIGNAL ON SYNTAX NAME ERRPROC           /* Trap syntax erros      */
   SIGNAL ON NOVALUE NAME ERRPROC          /* Trap unititialized vars*/
/* SQLTRACE='TRACE'                          RXSQL TRACE MODE       */
                                           /*                        */
/* ********************************************************************* */
/* Connect to a DB2 subsystem                                            */
/* ********************************************************************* */
                                           /*                        */
/* *** Connect to a DB2 subsystem                                        */
                                           /*                        */
   CALL "RXSQL" "CONNECT DSN"              /*                        */
   IF SQLCODE<>0 THEN DO                   /*                        */
      SAY 'CONNECT FAILED RC='SQLCODE      /*                        */
      SAY SQLMSG                           /*                        */
      EXIT 8                               /*                        */
   END                                     /*                        */
                                           /*                        */
   ADDRESS RXSQL                           /* DEFAULT HOST CMD ENV   */
   DBNAME='DSNDB06'                        /*                        */
                                           /*                        */
   OK='0';"RXSQL DECLARE C1 CURSOR FOR",   /*                        */
          "SELECT NAME, NACTIVE",
          "FROM SYSIBM.SYSTABLESPACE",
          "WHERE DBNAME = ?"
                                           /*                        */
   OK='0';"RXSQL DECLARE C2 CURSOR FOR",   /*                        */
          "SELECT NAME, CREATOR, CARD, NPAGES, PCTPAGES",
          "FROM SYSIBM.SYSTABLES",
          "WHERE DBNAME = ? AND TSNAME = ?"
                                           /*                        */
   OK='0';"RXSQL OPEN C1 USING :DBNAME"    /*                        */
                                           /*                        */
```

Figure 64: Sample RXSQL Program

```
   DO WHILE RXSQL("FETCH C1 INTO :TSNAME, :FLD2")=0
     SAY 'TABLE_SPACE_NAME='TSNAME 'NACTIVE='FLD2
     OK='0';"RXSQL OPEN C2 USING :DBNAME, :TSNAME"
                                        /*                        */
     DO WHILE RXSQL("FETCH C2 INTO :TBNAME, :TBCREATOR,",
                    ":CARD, :NPAGES, :PCTPAGES")=0
       SAY '> TABLE_NAME='TBNAME 'CREATOR='TBCREATOR,
           'CARD='CARD 'NPAGES='NPAGES 'PCTPAGES='PCTPAGES
     END                                /*                        */
                                        /*                        */
     OK='0';"RXSQL CLOSE C2"            /*                        */
   END                                  /*                        */
   OK='0';"RXSQL CLOSE C1"              /*                        */
                                        /*                        */
   CALL CLEANUP                         /* Clean up and exit      */
                                        /*                        */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Clean up and exit                                              */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
CLEANUP:

   CALL "RXSQL" "DISCONNECT"            /* Disconnect from DB2    */
                                        /*                        */
/* *** Turn off error capture                                     */
                                        /*                        */

   SIGNAL OFF SYNTAX                    /* Don't trap syntax      */
   SIGNAL OFF NOVALUE                   /* Dont trap uninit'd var */
   CALL OFF ERROR                       /* don't trap Cmd errors  */
                                        /*                        */
   EXIT 0                               /*                        */
                                        /*                        */
RETURN
```

Figure 64: Sample RXSQL Program

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* SUBROUTINE FOR ERROR HANDLING                                       */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
ERRPROC:
   SAVERC=RC
   IF (CONDITION('C')='ERROR' & SYMBOL('OK')='VAR') THEN
   DO
     IF WORDPOS(SAVERC,OK)>0 | OK='*' THEN DO
       OK=
       RETURN
     END
   END

   IF SYMBOL('RECURSIVE')='VAR' THEN,
      IF RECURSIVE='YES' THEN CALL CLEANUP
   RECURSIVE='YES'

   IF CONDITION('C')='SYNTAX' THEN DO
     IF ERRORTEXT(SAVERC)<>"" THEN
        ERRTEXT=ERRORTEXT(SAVERC)
   END
   ELSE ERRTEXT=CONDITION('I')
   SRCLINE='LINE='SIGL SOURCELINE(SIGL)
   IF SYMBOL('ZERRLM')<>'VAR' THEN ZERRLM=
   IF SYMBOL('ZERRSM')<>'VAR' THEN ZERRSM=
   IF ZERRLM='' THEN ZERRLM=CONDITION('D')
   IF ZERRSM='' THEN ZERRSM=CONDITION('C') 'LINE='SIGL
   SAY '   ERROR TEXT='ERRTEXT
   SAY '  SOURCE LINE='SRCLINE
   SAY 'SHORT MESSAGE='ZERRSM
   SAY ' LONG MESSAGE='ZERRLM
   IF CONDITION('C')='ERROR' THEN DO
     IF SYMBOL('SQLCODE')='VAR' THEN SAY '     SQL CODE='SQLCODE
     IF SYMBOL('SQLMSG')='VAR' THEN SAY '  SQL MESSAGE='SQLMSG
   END
   SAY 'Do you want to continue (YES/NO) ===>'
   PULL CONT
   RECURSIVE='NO'

   IF CONT<>'YES' THEN CALL CLEANUP
```

Figure 64: Sample RXSQL Program

SQL Statements

Connect to a DB2 Subsystem (CONNECT)

The **CONNECT** command initializes the RXSQL host command environment and connects to a DB2 subsystem. **CONNECT** may only be invoked via the **CALL "RXSQL"** external routine function, or as a function.

Command Format

```
RXSQL CONNECT
        subsys_name
```

Parameters

subsys_name	Specifies the name of the DB2 subsystem. RXSQL may only connect to 1 DB2 subsystem at a time.

Special Variables

Upon completion, RXSQL will set the following REXX variables to new values.

SQLCODE Return code from last operation.
`0` Successful
`-3` Syntax error
For other, see **SQLCODE** and **SQLMSG** for further information in *Appendix B: RXSQL Return Codes* on page 167.

RESULT Same as **SQLCODE** when invoked as external subroutine (**CALL "RXVSQL"**).

SQLMSG Descriptive message.

Disconnect from a DB2 Subsystem (DISCONNECT)

DISCONNECT terminates the RXSQL host command environment and disconnects from the DB2 subsystem. **DISCONNECT** may only be invoked via the **CALL "RXSQL"** external function, or as a function.

Command Format

```
RXSQL DISCONNECT
```

Special Variables

Upon completion, RXSQL will set the following REXX variables to new values.

SQLCODE	Return code from last operation. 0 Successful -3 Syntax error For other, see **SQLCODE** and **SQLMSG** for further information in *Appendix B: RXSQL Return Codes* on page 167.
RESULT	Same as **SQLCODE** when invoked as external subroutine (**CALL "RXVSQL"**).
SQLMSG	Descriptive message.

Declare Cursor

Up to one hundred (100) cursors may be declared and used at any one time. Cursor names may be any name up to 32 characters in length.

```
"RXSQL DECLARE C1 CURSOR FOR
   SELECT DEPTNO, DEPTNAME, MGRNO, FROM DSN850.DEPT
   WHERE ADMRDEPT = 'A11'"

"RXSQL OPEN C1"

DO WHILE RXSQL("FETCH C1 INTO :DNUM, :DNAME, :MNUM")
  . . .
  (process results)
  . . .
END

"RXSQL CLOSE C1"
```

Figure 65: Declare Cursor

The **FETCH** statement fetches rows from `DSN888.DEPT` until there are no more rows. The row columns: `DEPTNO/DEPTNAME/MGRNO`, are fetched into variables: `DNUM/DNAME/MNUM` respectively.

Declare Statement Names

Up to one hundred (100) statement names may be declared and used at a given time. Statement names may be any name up to 32 characters in length.

```
 SELECT_STATEMENT="SELECT DEPTO, DEPTNAME, MGRNO FROM",
"DSN888.DEPT WHERE ADMRDEPT = 'A11'"

 "RXSQL DECLARE STMT1"

 "RXSQL DECLARE CSR1 WITH HOLD FOR STMT1"

 "RXSQL PREPARE STMT FROM :SELECT_STATEMENT"

 "RXSQL OPEN CSR1"

 DO WHILE SQLCODE=0
   "RXSQL FETCH CSR1"
   . . .
   (process results)
   . . .
 END

 "RXSQL CLOSE CSR1"
```

Figure 66: Declare Statement Names

The **FETCH** statement fetches rows from `DSN888.DEPT` until there are no more rows. The REXX variables, `DEPTNO/DEPTNAME/MGRNO`, will be assigned as each row is retrieved.

EXECUTE USING

EXECUTE USING executes a **SQL DELETE**, **INSERT** or **UPDATE** statement, optionally supplying values for the parameter markers. The SQL statement must have been previously defined using **DECLARE** and **PREPARE**. The standard **SQL USING** clause allows indicator and host variables or a **SQLDA** to supply the defined values. RXSQL does not support indicator variables or a **SQLDA**, but does allow host variables and the key word **NULL**. RXSQL also allows the **FILE** clause to specify that the statement is to be executed once for each result row in a sequential data set or UNIX Systems Services file. The data set or file must have been written using the MAX/REXX **SELECT INTO FILE** statement or the MAX DB2/UTIL **CUT** or **UNLOAD** functions.

Command Format

```
RXSQL EXECUTE statement_name
          [ USING { {NULL | :variable_name},... |
          FILE({ddname | file_handle})
          [BYPASS(sqlcode,...)                    ]
          [MESSAGES(SAY | OFF | QUEUE)] }         ]
```

Parameters

statement_name	Specifies the name of a previously defined SQL statement.
NULL	Indicates that the corresponding parameter marker is to be replaced with a null value.
variable_name	Specifies the name of a REXX variable which contains the value to be assigned to the corresponding parameter marker.
ddname	Specifies the **ddname** of a sequential data set from which values will be read.
file_handle	Specifies the number of an open UNIX Systems Services file handle from which value will be read.
sqlcode	Specifies a list of one or more numbers identifying **SQLCODE** values which will not cause the **EXECUTE USING FILE** statement to halt. The numbers can be positive (warning values) or negative (error values). The keyword **WARN** is to indicate that all warnings are to be bypassed in addition to any listed error codes.
SAY | OFF | QUEUE	Specifies the disposition of messages that are generated by each execution of the statement, but which are bypassed due to the **BYPASS** clause. **SAY** causes the messages to be written using the REXX **SAY** facility. **OFF** causes the messages to be suppressed. **QUEUE** causes the messages to be written to the REXX external data queue. If the **MESSAGES** clause is omitted, the default is **SAY**.

Special Variables

In addition to **SQLCODE**, **SQLMSG** and the other special variables, successful completion of an **EXECUTE USING FILE** will cause the following variables to be set:

SQLERRD.1	Set with the number of rows read from the data set or file.
SQLERRD.2	Set with the number of times the statement was successfully executed.
SQLERRD.3	Set with the number of SQL rows deleted, inserted or updated.

SELECT INTO

SELECT INTO retrieves an entire result set in a single operation. The standard **SQL INTO** clause of the **SELECT** statement listing host variables is fully supported for single-row result set, including the use of indicator variables. RXSQL also supports extensions to the **INTO** clause allowing multiple result rows to be retrieved into an ISPF table, REXX stem variable arrays, an sequential data set or UNIX Systems Services file.

Command Format

```
RXSQL SELECT ...
        INTO { :variable_name [:indicator_variable],... |
        {ISPTABLE(table_name) | STEM() |
            FILE({ddname | file_handle}) [FORMAT(format) [CODEPAGE(codepage)] ] }
        [LIMIT(limit_count)                                 ]
        [SKIP(skip_count)                                   ]
        [EVERY(every_count)                                 ]
        [IGNORE(100)                                        ] } }
        FROM ...
```

Parameters

variable_name	Specifies the name of a REXX variable which will be set to the column value.
indicator_variable	Specifies the name of a REXX variable which will be set to '0' if the column value is not null or '-1' if the column value is null.
table_name	Specifies the name of an ISPF table into which the result set will be written.
ddname	Specifies the **ddname** of a sequential data set into which the result set will be written.
file_handle	Specifies the number of an open UNIX Systems Services file handle to which the result set will be written.
format	Specifies the name of a Data Transformation template defining the format of the data to be written to the data set or file handle. Predefined formats include XML, CSV, TAB, and RAW. Additional formats may be defined in the MAXDFLTS module. If the **FORMAT** clause is omitted, the result set will be written in an internal format suitable for the MAX/REXX **EXECUTE USING FILE** statement or the MAX DB2/UTIL **PASTE** or **LOAD** facilities.
codepage	Specifies a code page number from 1 to 65535, which overrides the default output code page for the named **FORMAT**.
limit_count	Specifies a number from 1 to 999999999, which limits the number of result rows returned or written. If the **LIMIT** clause is not specified, all result rows will be returned or written.

skip_count	Specifies a number from `1` to `999999999`, which causes the first **skip_count** rows of the result set to be skipped before the first row is returned or written. If the **SKIP** clause is not specified, no initial result set rows will be skipped.
every_count	Specifies a number from `1` to `999999999`, which causes only every other **every_count** rows to be written. If the **EVERY** clause is not specified or **EVERY(1)** is specified, every row (after **skip_count**) will be returned or written.
IGNORE(100)	Causes RXSQL to ignore SQL warning code `100` that is normally returned when no result set rows are returned or written.

Special Variables

In addition to **SQLCODE**, **SQLMSG** and the other special variables, successful completion of a **SELECT INTO** with a **STEM**, **ISPTABLE** or **FILE** clause will cause the following variables to be set:

SQLERRD.1	Set to the number of rows fetched.
SQLERRD.2	Set with the number of rows returned or written.

SELECT INTO ISPTABLE

DB2 column data may be loaded directly into a dynamically prepared ISPF table. The column names being selected must be valid ISPF variable names.

```
/* REXX  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* DOC: RXSQL3   - Lists all tables (SELECT INTO ISPTABLE)           */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
   ARG SUBSYS                           /* GET PASSED PARAM       */
   IF SUBSYS="" THEN DO                 /* Subsys specified ?     */
     SAY 'DB2 Subsystem not specified - defaulting to DSN'
     SUBSYS='DSN'                       /*                        */
   END                                  /*                        */
                                        /*                        */
/* ********************************************************************* */
/* Connect to a DB2 subsystem                                        */
/* ********************************************************************* */
                                        /*                        */
   CALL "RXSQL" "CONNECT" SUBSYS        /*                        */
   IF SQLCODE<>0 THEN DO                /*                        */
      SAY 'RXSQL CONNECT Failed' SUBSYS /*                        */
      EXIT 8                            /*                        */
   END                                  /*                        */
                                        /*                        */
```

Figure 67: RXSQL program using **SELECT** into an ISPF table

```
/* ************************************************************** */
/* List name, dbname, tsname, from sysibm.systables               */
/* ************************************************************** */
                                       /*                         */
   CALL "RXSQL" "SELECT NAME, DBNAME, TSNAME",
                "INTO ISPTABLE(MYNAME)",
                "FROM SYSIBM.SYSTABLES",
                "ORDER BY DBNAME, TSNAME, NAME"
   IF SQLCODE<>0 THEN CALL CLEANUP     /*                         */
                                       /*                         */
   ADDRESS ISPEXEC "TBTOP MYNAME"      /* Top of table            */
                                       /*                         */
   DO UNTIL RC<>0                      /* Loop until problem/EOF  */
     ADDRESS ISPEXEC "TBSKIP MYNAME"   /*  Get next row in table  */
     IF RC<>0 THEN ITERATE             /*                         */
     SAY 'DBNAME='SUBSTR(DBNAME,1,12,' '), /*  List columns         */
         'TSNAME='SUBSTR(TSNAME,1,12,' '), /*                      */
         'NAME='NAME                   /*                         */
   END                                 /*                         */
                                       /*                         */
   ADDRESS ISPEXEC "TBEND MYNAME"      /* Top of table            */
                                       /*                         */
   CALL CLEANUP                        /* Clean up and exit       */
                                       /*                         */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Clean up and exit                                              */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
CLEANUP:

   IF SQLCODE<>0 THEN DO               /*                         */
      SAY 'RXSQL failed RC='SQLCODE    /*                         */
      SAY SQLMSG                       /*                         */
   END                                 /*                         */
                                       /*                         */
   CALL "RXSQL" "DISCONNECT"           /* Disconnect from DB2     */
                                       /*                         */
   EXIT 0                              /*                         */
```

Figure 67: RXSQL program using **SELECT** into an ISPF table

SELECT INTO STEM

DB2 column data may be loaded directly into a variable stem array. Each column name will be appended with a '.' and a row number (for example: `column_name.1`, `column_name.2`, ...). In addition, a `column_name.0` variable will contain the total number of occurrences in the stem array.

```
/* REXX  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* DOC: RXSQL4   - Lists all tables (SELECT INTO STEM())              */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
   ARG SUBSYS                          /* GET PASSED PARAM        */
   IF SUBSYS="" THEN DO                /* Subsys specified ?      */
     SAY 'DB2 Subsystem not specified - defaulting to DSN'
     SUBSYS='DSN'                      /*                         */
   END                                 /*                         */
                                       /*                         */
/* ********************************************************************** */
/* Connect to a DB2 subsystem                                         */
/* ********************************************************************** */
                                       /*                         */
   CALL "RXSQL" "CONNECT" SUBSYS       /*                         */
   IF SQLCODE<>0 THEN DO               /*                         */
      SAY 'RXSQL CONNECT Failed' SUBSYS /*                         */
      EXIT 8                           /*                         */
   END                                 /*                         */
                                       /*                         */
/* ********************************************************************** */
/* List name, dbname, tsname, from sysibm.systables                   */
/* ********************************************************************** */
                                       /*                         */
   CALL "RXSQL" "SELECT NAME, DBNAME, TSNAME",
                "INTO STEM()",
                "FROM SYSIBM.SYSTABLES",
                "ORDER BY DBNAME, TSNAME, NAME"
   IF SQLCODE<>0 THEN CALL CLEANUP     /*                         */
                                       /*                         */
   DO NDX=1 BY 1 TO DBNAME.0           /* Loop until problem/EOF */
     SAY 'NDX='NDX
     SAY 'DBNAME='SUBSTR(DBNAME.NDX,1,12,' '), /*  List columns     */
         'TSNAME='SUBSTR(TSNAME.NDX,1,12,' '), /*                  */
         'NAME='NAME.NDX               /*                         */
   END                                 /*                         */
                                       /*                         */
   CALL CLEANUP                        /* Clean up and exit       */
                                       /*                         */
```

Figure 68: RXSQL program using **SELECT** into stem variables

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Clean up and exit                                                   */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
CLEANUP:

   IF SQLCODE<>0 THEN DO                  /*                        */
      SAY 'RXSQL failed RC='SQLCODE       /*                        */
      SAY SQLMSG                          /*                        */
   END                                    /*                        */
                                          /*                        */
   CALL "RXSQL" "DISCONNECT"              /* Disconnect from DB2    */
                                          /*                        */
   EXIT 0                                 /*                        */
```

Figure 68: RXSQL program using **SELECT** into stem variables

DB2 columns that are not defined as **NOT NULL** have an additional stem variable set for each row to indicate whether the value is null. The names of these indicator variables will be the column name appended with a '`?.`' and a row number (for example: `column_name?.1`, `column_name?.2`, …). The `column_name?.0` variable is not set. The indicator variables will not be set for DB2 columns defined as **NOT NULL**.

Requesting ISPF Services

With RXSQL, you do not have to switch host command environments to invoke ISPF services. Precede any ISPF service with "`ISPEXEC`" and RXSQL will invoke the ISPF service in its own host command environment.

The return codes are the same as if the ISPF service was invoked without RXSQL.

```
  ADDRESS RXSQL                       /* set default environment  */
                                      /* display ISPF panel       */
  "ISPEXEC DISPLAY PANEL(DENTRY)"
 IF RC<>0 THEN ....
```

Figure 69: Requesting ISPF Services

Requesting PDF Edit Services

With RXSQL, it is not necessary to switch host command environments to invoke PDF Edit services. Precede any PDF Edit service with "ISREDIT" and RXSQL will invoke the PDF Edit service in its own host command environment.

The return codes are the same as if the PDF Edit service was invoked without RXSQL.

```
ADDRESS RXSQL                          /* set default environment  */
                                       /* begin Edit Macro         */
"ISREDIT MACRO (OPT) NOPROCESS"
IF RC<>0 THEN ....
```

Figure 70: Requesting PDF Edit Services

CHAPTER 5: MVS PROCESSING

Summary

This MVS processing feature of MAX/REXX provides both access to some often needed MVS facilities and common routines required when processing data.

MVS Facilities Statement	
ALLOCATE	Allocate a file.
CONNECT	Connect to RXMVS, establish as a host command environment.
DEQ	Issue a **DEQ**ue of a resource.
DISCONNECT	Disconnect from RXMVS, terminate the host command environment.
ENQ	Issue an **ENQ**ue on a resource.
FREE	De-allocate a file.
LINK	Link to another program.
LISTCAT	Provide catalog information.
LISTVTOC	Return volume information.

Common Processing Routines	
ASSIGN	Dynamic assignment of a value to a variable.
SORT	Sort an array of data.

Date Processing Routines	
CALCDATE	Compute a new date (Gregorian).
CALCGDIF	Compute number of days between two Gregorian dates.
CALCJDIF	Compute number of days between two Julian dates.
CALCJUL	Compute a new date (Julian).
DATE2JUL	Convert Gregorian date to Julian.
JUL2DATE	Convert Julian date to Gregorian.

Character Processing Routines	
CONVERT	Convert characters from one code page to another.
CPCLASS	Determine if a code page is EBCDIC, ASCII or Unicode.
CPLIST	List the supported code pages.
CPNAME	Return a code page description.
DECODE64	Decode base-64 encoded data.
ENCODE64	Encode data using base-64.
ISALNUM	Test if characters are alphanumeric.
ISALPHA	Test if characters are alphabetic.
ISBLANK	Test if characters are blanks.
ISCNTRL	Test if characters are control characters.
ISDIGIT	Test if characters are digits.
ISGRAPH	Test if characters are non-blank printing characters.
ISLOWER	Test if characters are lowercase letters.
ISPRINT	Test if characters are printing characters.
ISPUNCT	Test if characters are punctuation.
ISSPACE	Test if characters are spacing characters.
ISUPPER	Test if characters are uppercase letters.
ISXDIGIT	Test if characters are hexadecimal digits.
TOLOWER	Convert characters to lowercase.
TOUPPER	Convert characters to uppercase.

Special Variables

MVSCODE

The special variable **MVSCODE** will always contain the return code following any invocation of RXMVS. This variable is available whether RXMVS was invoked via the host command environment (**ADDRESS RXMVS**), an external routine (**CALL "RXMVS"**), or as a function (**RXMVS (...)**). This variable may be used to test for the successful completion of a RXMVS statement regardless of how RXMVS was invoked. This facilitates the use of a common error routine to handle all RXMVS errors.

```
CALL "RXMVS" "ASSIGN INGO (VAR01) FROM (VAR_NAME."VAR#")"
IF MVSCODE<>0 THEN DO
   SAY 'MVS RETURN CODE='MVSCODE
   SAY 'RC DESCRIPTION ='MVSMSG
END
```

Figure 71: **MVSCODE**

MVSMSG

The special variable **MVSMSG** will always contain a descriptive message following any invocation of RXMVS. This variable is available whether RXMVS was invoked via the host command environment (**ADDRESS RXMVS**), an external routine (**CALL "RXMVS")** or as a function (**RXMVS (...)**). This variable contains descriptive text that can be used to further assist in determining the problem.

```
CALL "RXMVS" "SORT FROM(VAR_ARRAY.) INTO(VAR_ARRAY.) DESCENDING"
IF MVSCODE<>0 THEN DO
   SAY 'ERROR FROM SORT='MVSCODE
   SAY 'ERROR DESCRIPTION='MVSMSG
END
```

Figure 72: **MVSMSG**

MVSTRACE

When the special variable **MVSTRACE** is assigned to the value `TRACE`, RXMVS will write the resolved statement to `SYSTSPRT`. This special trace uses the REXX **SAY** function and may be used in any REXX environment. Tracing may be stopped by assigning **MVSTRACE** to any value other than `TRACE`. This tracing will occur whether RXMVS was invoked via the host command environment (**ADDRESS RXMVS**), an external routine (**CALL "RXMVS"**), or as a function (**RXMVS (...)**).

```
MVSTRACE="TRACE"
```

Figure 73: **MVSTRACE**

MVS Facilities Statements

The following statements made available by MAX/REXX, allow the REXX application to exploit MVS functions not otherwise usable by a REXX application.

Link to Another Program (LINK)

The **LINK** statement makes it possible for the REXX program to pass control to another program. Control is passed using the standard IBM linkage conventions, and parameter data may be passed to the program.

Command Format

```
RXMVS LINK
          PGM(pgmname)
          [ PARM(parmdata)                    ]
```

Parameters

pgmname	Specifies the name of the program that is the object of the **LINK**. In a TSO environment, no authorized programs except **IEBCOPY** can be invoked.
parmdata	Parameter data to be passed to the program with the **LINK**. The data can be up to a maximum of 100 bytes in length. If the data starts with a colon and ends with a period, the data is recognized as the name of a stem variable, for example: **PARM(:STEM.)**. The `.0` stem variable must contain a single digit with the number of parameters (1, 2 or 3). The `.1`, `.2` and `.3` stem variables must contain the parameters to be passed to the program. Each of the parameters can be up to a maximum of 256 bytes in length.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from last operation.

`0`	Successful
`-3`	Syntax error
`12`	Error occurred during the **LINK**

RESULT Same as **MVSCODE** when invoked as external subroutine (**CALL "RXVMVS"**).

MVSMSG Descriptive message.

```
                                  /* link to a batch program     */
RC=RXMVS("LINK PGM(PAY5) PARM(DATE=2000/01/01)")
```

Figure 74: Link to Program '**PAY5**'

ENQue a Resource (ENQ)

The **ENQ** command allows the REXX program to gain control over a resource. The standard MVS **ENQ**ue facility is used, allowing the MAX/REXX program to participate in system wide serialization of resources.

Command Format

```
RXMVS ENQ
          RESOURCE(resourcename)
          [ QNAME(qname)                                ]
          [ SCOPE(STEP|SYSTEM|SYSTEMS)                  ]
          [ CONTROL(E|S)                                ]
          [ RETURN(NONE|CHNG|HAVE|TEST|USE))            ]
```

The parameters for this command are briefly explained in this manual, but for additional information on these parameters, refer to the appropriate IBM manual.

Parameters

resourcename	This is the name of the resource name on which to issue the **ENQ**ue. This name can be up to 255 bytes in length. This name must match, exactly, the resource name used by any other task that wants to serialize the use of this resource.
qname	This is an 8-character field that contains a second value to further identify the resource that is to be the object of the **ENQ**ue. This field is optional, if not specified, RXMVS uses a value of `MAXREXX1` for the **qname**. If you need to serialize the resource with an application that is not written in MAX/REXX, it is recommended that you choose the **qname** and not allow it to default.
STEP|SYSTEM|SYSTEMS	This value specifies the scope of the resource to be used. **STEP** (default) limits the scope of the **ENQ**ue to be within the address space. **SYSTEM** expands the scope to programs running in other address spaces on the same system. **SYSTEMS** further expand the scope to be shared across systems. The scope is a critical part of the **ENQ**ue. If the **resourcename** and **qname** are the same but the scope is different, it is considered as a different resource.

E|S	This value specifies the request as an **E**xclusive (the default) or **S**hared control of the resource. If the resource is modified while under control of the task, the request must be for exclusive control; if the resource is not modified, the request should be for shared control.
NONE|CHNG|HAVE|TEST|USE	This value specifies the type of request for **ENQ**ue being requested. **NONE** (default) requests control of the resource unconditionally. **CHNG** specifies the status of the resource is to be changed from shared to exclusive. **HAVE** specifies that control is requested only if a request has not been previously made for the same task. **TEST** is available to test the status, but does not request control. **USE** requests that control be assigned only if the resource is immediately available.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from last operation.

0	Successful
-3	Syntax error
4	**RET=TEST|USE**: resource not immediately available. **RET=CHNG**: status cannot be changed.
8	**RET=TEST|USE|HAVE**: task has previously requested control, as control.
14	Task has previously made request for control, does not have control.
12	Unable to process statement.
18	**RET=HAVE|USE**: limit for concurrent requests has been reached.

RESULT Same as **MVSCODE** when invoked as external subroutine (**CALL "RXMVS"**).

MVSMSG Descriptive message.

```
/* *** this example will serialize on a DDNAME and MEMBER        */
/* ***      prior to update                                      */

  DDNAME='SYSUT1'
  MEM_NAME='JOBONE'
  RC=RXMVS("ENQ RESOURCE("DDNAME") QNAME("MEM_NAME") CONTROL(E)",
           "SCOPE(SYSTEM) RETURN(USE)")
```

Figure 75: **ENQ**

DEQue a Resource (DEQ)

The **DEQ** command is available to allow the REXX program to release control of a resource that it has previously obtained control through an **ENQ** command. The standard MVS **DEQ**ue facility is used, allowing the MAX/REXX program to participate in system wide serialization of resources.

Command Format

```
RXMVS DEQ
          RESOURCE(resourcename)
          [ QNAME(qname)                              ]
          [ SCOPE(STEP|SYSTEM|SYSTEMS)                ]
          [ RETURN(NONE|HAVE)                         ]
```

The parameters for this command are briefly explained in this manual, but for additional information on these parameters, refer to the appropriate IBM manual.

Parameters

resourcename	This is the name of the resource name on which to issue the **DEQ**ue. This name can be up to 255 bytes in length. This name must match, exactly, the resource name used by any other task that wants to serialize the use of this resource. In addition, this name must match that used in the ENQ request for control of the resource.
qname	This is an 8 character field that contains a second value to further identify the resource that is to be the object of the **DEQ**ue. *This field is optional*. However, if it was used in the **ENQ** request, then it must be used in the **DEQ**, and must match the name used in the **ENQ** exactly.
STEP|SYSTEM|SYSTEMS	This value specifies the scope of the resource to be used. Refer to **ENQ** for a description of the terms. This is considered part of the name used in obtaining control of the resource, and therefore, must match that used in the **ENQ** request.
NONE|HAVE	This value specifies conditions of the release of the resource. **NONE** is an unconditional request for release of the resource. If the task does not have control of the resource and this type of request is issued, the task will abend. **HAVE** is a conditional request in that it will cause a release of control only if the task has the resource assigned to it. If the resource is not assigned, the task will not abend.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from last operation.

0	Successful
-3	Syntax error
4	Resource has been requested for the task, but task has not been assigned control
8	Control of the resource has been requested or was previously released
12	Unable to process statement.

RESULT Same as **MVSCODE** when invoked as external subroutine (**CALL "RXMVS"**).

MVSMSG Descriptive message.

```
/* *** DEQUE the resource that was enqued prior to update          */

  RC=RXMVS("DEQ RESOURCE("DDNAME") QNAME("MEM_NAME")",
           "SCOPE(SYSTEM) RETURN(HAVE)")

/* Note:  in the above example, all parameters except RETURN       */
/*        must match exactly with those used in the ENQ instruction */
```

Figure 76: **DEQ**

Return VTOC Information (LISTVTOC)

The **LISTVTOC** command allows a REXX program to access all of format 1 DSCB information in the VTOC.

Command Format

RXMVS LISTVTOC
VOL(volser)
[INTO(variable name)]

Parameters

volser	Specifies the volumn serial number of the volume that contains the format 1 DSCB's to be returned.
variable name	Specifies the base variable name into which the format 1 data will be returned. Each format 1 will be returned in a unique variable. If this parameter is not supplied, the variable name will be set to the volume serial number and used as a stem variable. The total number of format 1 DSCBs returned will be returned in the zero (0) variation of the variable. Pass a stem variable as this value to allow the REXX program to easily iterate through a loop, processing each variable value that is returned. See example on following page for information on how to access this data.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from last operation.

`0` Successful
`-3` Syntax error
`12` Unable to process statement.

RESULT Same as **MVSCODE** when invoked as external subroutine (**CALL "RXMVS"**).

MVSMSG Descriptive message.

Note: The following example is available for demonstration purposes. See the installation EXECS library member MAXXREXX1

```
/*  REXX                                                          */
/*  DOC:  Sample to list all datasets on a volume with creation date */
/*                                                                */
/*  This sample MAX/REXX program uses the following features of the  */
/*  product:                                                      */
/*  RXVSAM:  OPEN, FORMAT, CLOSE                                  */
/*  RXMVS:   LISTVTOC                                             */
/*                                                                */
/*  To run this example do the following:                         */
/*  1.  copy the member DSCB1 from installation EXECS library to  */
/*      one of your own PDS files                                 */
/*  2.  set the value of the VOLINFO_VOLUME variable to a volume  */
/*      serial number of a volume on your system                  */
/*  3.  set the value of the DSNNAME variable to the name of the  */
/*      library where you copied the member in number 1 above     */
                                        /*                        */
   VOLINFO_VOLUME='nnnnnn'              /* change this value !!!  */
   DSNNAME='MXS.TEST.PDS'               /* change this value !!!  */
                                        /*                        */
/* *** FETCH FMT1'S FROM VTOC                                     */
                                        /*                        */
   IF RXMVS("LISTVTOC VOL("VOLINFO_VOLUME") INTO(FMT1.)")<>0,
       THEN DO                          /*                        */
       SAY MVSCODE ',' MVSMSG           /* error in LISTVTOC      */
       SAY 'Verify that you set the variable: VOLINFO_VOLUME'
       EXIT 8                           /*                        */
   END                                  /*                        */
                                        /*                        */
/* *** use MAX/REXX to OPEN the copybook file                     */
                                        /*                        */
   IF RXVSAM("OPEN FILE(DSCB1) PLICOPY("DSNNAME"(DSCB1)) DUMMY")<>0 THEN
   DO                                   /* problem                */
      SAY VSAMCODE ',' VSAMMSG          /* RXVSAM problem         */
      SAY 'Verify that you set the correct PDS name to find copybook'
      EXIT 8                            /* Return immediatly      */
   END                                  /*                        */
                                        /*                        */
```

Figure 77: Program to list all data sets on volume (**LISTVTOC**) along with creation date

```
/* *** SHOW INFORMATION FOUND IN THE FORMAT 1                        */
                                        /*                           */
   DO NUM=1 TO FMT1.0                   /* loop thru all returned */
     IF SUBSTR(FMT1.NUM,45,1)<>'F1'X THEN ITERATE /* validate format1*/
                                        /*                           */
/* *** use RXVSAM to format the information into variable fields     */
                                        /*                           */
     IF RXVSAM("FORMAT FILE(DSCB1) FROM(FMT1."NUM")")>0 THEN
        SAY VSAMCODE ',' VSAMMSG        /* RXVSAM problem         */
                                        /*                           */
     SAY 'NAME='DS1DSNAM                /* display the dsname     */
     YY=X2D(C2X(SUBSTR(DS1CREDT,1,1)))
     DDD=RIGHT(X2D(C2X(SUBSTR(DS1CREDT,2,2))),3,'0')
     DATE=YY'/'||DDD
     SAY 'CREATE='DATE                  /* display creation date  */
   END                                  /*                           */
                                        /*                           */
   CALL "RXVSAM" "CLOSE FILE(DSCB1)"    /* close the file         */
                                        /*                           */
EXIT 0                                  /*                           */
                                        /*                           */
```

Figure 77: Program to list all data sets on volume (**LISTVTOC**) along with creation date

```
NAME=MXS.TEST.KSDS173.DATA
 CREATE=98/344
 NAME=MXS.TEST.KSDS173.INDEX
 CREATE=98/344
 NAME=MXS.TEST.KSDS2VOL.DATA
 CREATE=98/064
 NAME=MXS.TEST.MAAPDS2
 CREATE=99/219
 NAME=MXS.TEST.MARFILE
 CREATE=100/342
 NAME=MXS.TEST.MARLIMT
 CREATE=100/347
 NAME=MXS.TEST.MAR80
 CREATE=100/350
 NAME=MXS.TEST.MXRXTEST.KSDS.DATA
 CREATE=98/177
 NAME=MXS.TEST.MXRXTEST.KSDS.INDEX
 CREATE=98/177
```

Figure 78: Sample Output from **LISTVTOC** program

Return Data Set Information from Catalog (LISTCAT)

The **LISTCAT** command allows a REXX program to obtain a list of data sets by passing a high level qualifier. Some limited additional information is returned along with the data set name.

Command Format

```
RXMVS LISTCAT
          QUAL (qualifier)
          [ CATALOG (catalog_name)                ]
          [ PASSWORD (password)                   ]
          [ INTO(varname)                         ]
```

Parameters

qualifier	Specify the high level qualifier for the data sets to be returned. This qualifier can be specified in multiple formats. For example, you have the following list of data sets: 1. MXS.TEST.SAM 2. MXS.TEST.SAMFIX 3. MXS.TEST.SAM.FILE Enter: `MXS.TEST.SAM*` as the qualifier. All three names will be returned. Enter: `MXS.TEST.SAM` as the qualifier. The #1 and # 3 data set names will be returned. Enter: `MXS.TEST.SAM.*` as the qualifier. File #3 is the only file returned. **Note:** MXS.TEST.SAM. is not a valid entry.
catalog_name	To restrict the data set names returned to those from a specific catalog, enter the catalog name here.
password	Enter the password for the catalog in the **CATALOG** parameter if the catalog is password protected.

varname	Specifies the base variable name into which the data set information will be returned. If the variable name is not specified, the name will be derived from the input qualifier. This method could cause an error if the qualifier cannot be used to build a valid variable name. The total number of data sets returned will be stored into the zero (0) variation of the variable. Pass a stem variable to allow the REXX program to easily iterate through a loop, processing each variable value that is returned. Format of the variable data that is returned: **pos** / **length** / **value** `1` / `1` / data set type `G` – AIX `N` – non VSAM `D` – Data (VSAM) `I` – Index (VSAM) `2` / `44` / data set name `46` / `1` / number of volumes `47` / `4` / device type `51` / `6` / vol ser `57` / `2` / vol sequence The final three fields are repeated for the number of volumes on which the data set is stored. If the data resides on two volumes, this information will be repeated in pos `59` for a length of `12`.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from last operation.
- `0` Successful
- `-3` Syntax error
- `12` Unable to process statement.

RESULT Same as **MVSCODE** when invoked as external subroutine (**CALL "RXMVS"**).

MVSMSG Descriptive message.

Note: The example on the next page is available for demonstration purposes. See the installation EXECS library member MAXXREXX1.

```
/*  REXX                                                          */
/*  DOC:  Sample to list dataset information from the catalog     */
/*                                                                */
/*  This sample MAX/REXX program uses the following features of the  */
/*  product:                                                      */
/*  RXMVS:   LISTCAT                                              */
/*                                                                */
/*  To run this example do the following:                         */
/*  1.  set the QUALIFIER to an appropriate high level qualifier for */
/*      your installation.                                        */
                                        /*                        */
   QUALIFIER='MXS.MXRX*'                /* change this value !!!! */
                                        /*                        */
/* *** retrieve catalog information                              */
                                        /*                        */
   IF RXMVS("LISTCAT QUAL("QUALIFIER") INTO(DSNS.)")<>0,
       THEN DO                          /*                        */
       SAY MVSCODE ',' MVSMSG           /* error in LISTVTOC      */
       EXIT 8                           /*                        */
   END                                  /*                        */
                                        /*                        */
/* *** show information found in the catalog                     */
                                        /*                        */
   DO NUM=1 TO DSNS.0                   /* loop thru all returned */
     DSNAME=SUBSTR(DSNS.NUM,2,44)       /*                        */
     SAY 'DATA SET NAME='DSNAME         /* dsname                 */
     DSNTYPE=SUBSTR(DSNS.NUM,1,1)       /*                        */
     SAY 'TYPE OF DATASET='DSNTYPE      /* type of dataset        */
     VOLCNT=C2X(SUBSTR(DSNS.NUM,46,1))  /*                        */
     SAY 'NUMBER OF VOLUMES='VOLCNT     /* number of volumes      */
     VOL_INFO=DELSTR(DSNS.NUM,1,46)     /*                        */
     DO I=1 TO VOLCNT                   /* repeat for each volume */
        VOLSER=SUBSTR(VOL_INFO,5,6)     /*                        */
        SAY 'VOLUME ' I'=' VOLSER       /*    volser              */
        VOLTYPE=C2X(SUBSTR(VOL_INFO,1,4))  /*                     */
        SAY 'VOLUME TYPE='VOLTYPE       /*    volume type         */
        VOLSEQ=C2X(SUBSTR(VOL_INFO,11,2))  /*                     */
        SAY 'VOLUME SEQ='VOLSEQ         /*    volume sequence     */
        VOL_INFO=DELSTR(VOL_INFO,1,12)  /*                        */
     END                                /*                        */
                                        /*                        */
   END                                  /*                        */
                                        /*                        */
   IF DSNS.0=0 THEN SAY 'No dsnames found for qualifier:' QUALIFIER
                                        /*                        */
EXIT 0                                  /*                        */
```

Figure 79: Sample program to list catalog information

```
DATA SET NAME=MXS.MXRXV240.MESSAGES
 TYPE OF DATASET=N
 NUMBER OF VOLUMES=01
 VOLUME  1= MAX003
 VOLUME TYPE=3010200E
 VOLUME SEQ=0001
 DATA SET NAME=MXS.MXRXV240.OBJECT
 TYPE OF DATASET=N
 NUMBER OF VOLUMES=01
 VOLUME  1= MAX003
 VOLUME TYPE=3010200E
 VOLUME SEQ=0001
```

Figure 80: Sample Output from LISTCAT program

Allocate a Data Set (ALLOCATE)

The **ALLOCATE** command is available for dynamic allocation of data sets.

Command Format

```
RXMVS ALLOCATE
          FILE(filename)
          [ DATASET(data set name)                ]
          [ NEW|OLD|MOD|SHR|                      ]
          [ DELETE|KEEP|CATLG|UNCATLG             ]
          [ SYSOUT( x )                           ]
          [ SPACE(nnnnnn|CYL|TRK)                 ]
          [ PRIMARY(nnnnn)                        ]
          [ SECND(nnnnn)                          ]
          [ DIRBLK(nnnnn)                         ]
          [ UNITNAME(unitname)                    ]
          [ DSORG(dsorg)                          ]
          [ RECFM(recfm)                          ]
          [ BLKSIZE(nnnnn)                        ]
          [ LRECL(nnnnn)                          ]
```

Parameters

filename	Specify a valid filename (one to eight characters) to be used to reference the data set to be allocated.
data set name	Specify the 1 to 44 character data set name. *This field is optional*. If this field is not specified, the file is assumed to be a SYSOUT data set.
NEW|OLD|MOD|SHR	Use one of these four options to specify the original disposition of the data set.
DELETE|KEEP| CATLG|UNCATG	Use one of these four options to specify the final disposition of the data set. This need only be specified to change the disposition of the data set from its input disposition.
X	Specifiy SYSOUT class for a SYSOUT data set.
nnnnn|CYL|TRK	Specify the type of allocation. If the numeric option is chosen, this is considered a block allocation. The number must be specified as a full 5 digits with the necessary leading zeroes.
nnnnn	This parameter is specified for several operands. Any time a numeric value is specified in this format, it must be a full 5 digits with the necessary leading zeroes. This parameter is used to specify primary and secondary allocation, number of directory blocks, block size, and logical record length.
unitname	Specify the unit name for allocation.
dsorg	Specify the data set organization. Valid values are: `PO`, `POU`, `DA`, `DAU`, `PS`, `PSU`.
recfm	Specify the record format of the data set to be allocated. Record formats supported are: Fixed, Variable, Undefined, Blocked, Spanned, Ascii Control, Machine Control, and Track Overflow.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from last operation.

`0`	Successful
`-3`	Syntax error
`12`	Unable to process statement.

RESULT Same as **MVSCODE** when invoked as external subroutine (**CALL "RXMVS"**).

MVSMSG Descriptive message.

Note: The following example is available for demonstration purposes. See the installation EXECS library member MAXXREXX1.

```
/*  REXX                                                          */
/*  DOC:  Sample to show file allocation/free                     */
/*                                                                */
/*  This sample MAX/REXX program uses the following features of the  */
/*  product:                                                      */
/*  RXMVS:   ALLOCATE, FREE                                       */
/*  RXVSAM:  OPEN, CLOSE, READNEXT, WRITE                         */
/*                                                                */
/*  To run this example do the following:                         */
/*  1.  set the DSN variable to a dataset name applicable to your    */
/*      environment.                                              */
/*  2.  set the TESTFILE variable to an appropriate ddname.       */
                                        /*                        */
   DSN='MXQUAL.TEST.FILE'               /* change this value !!!! */
   TESTFILE='SYSUT1'                    /* change this value !!!! */
                                        /*                        */
/* allocate the file as new,keep                                  */
                                        /*                        */
   IF "RXMVS"("ALLOCATE FILE("TESTFILE") DATASET("DSN")",
      "NEW CATLG SPACE(TRK) PRIMARY(00001) UNITNAME(SYSDA)",
      "DSORG(PS) RECFM(FB) BLKSIZE(06000) LRECL(00060)")<>0 THEN DO
        SAY 'UNABLE TO ALLOCATE'        /*                        */
        SAY 'Check the variable values set for DSN and TESTFILE'
        SAY 'ERROR CODE='MVSCODE', MESSAGE='MVSMSG    /*          */
        EXIT 8                          /*                        */
   END                                  /*                        */
                                        /*                        */
/* open the file for LOAD                                         */
                                        /*                        */
   IF "RXVSAM"("OPEN FILE("TESTFILE") LOAD")<>0 THEN DO
        SAY 'UNABLE TO OPEN FILE'       /*                        */
        SAY 'ERROR CODE='VSAMCODE', MESSAGE='VSAMMSG    /*         */
        EXIT 8                          /*                        */
   END                                  /*                        */
                                        /*                        */
```

Figure 81: Sample program using **RXMVS ALLOCATE**

```
/* write a series of records to the file                          */
                                        /*                        */
   DO NDX=1 TO 100                      /*                        */
      RCD="THIS IS TEST RCD NUMBER="NDX  /*                        */
      RCD=SUBSTR(RCD,1,60,' ')          /*                        */

IF "RXVSAM"("WRITE FILE("TESTFILE") FROM(RCD)")<>0 THEN LEAVE
   END                                  /*                        */
                                        /*                        */
/* close and free the file                                        */
                                        /*                        */

   RC="RXVSAM"("CLOSE FILE("TESTFILE")")  /* close the file       */
                                        /*                        */
   RC="RXMVS"("FREE FILE("TESTFILE")")  /* free the file          */
                                        /*                        */
/* reallocate the file as shr - specify delete when finished      */
                                        /*                        */
   IF "RXMVS"("ALLOCATE FILE("TESTFILE") DATASET("DSN")",
              "SHR DELETE")<>0 THEN DO  /*                        */
        SAY 'UNABLE TO ALLOCATE'        /*                        */
        SAY 'ERROR CODE='MVSCODE', MESSAGE='MVSMSG    /*           */
        EXIT 8                          /*                        */
   END                                  /*                        */
                                        /*                        */
/* open the file for input - sequential                           */
                                        /*                        */
   IF "RXVSAM"("OPEN FILE("TESTFILE") SEQ")<>0 THEN DO
        SAY 'UNABLE TO OPEN FILE'       /*                        */
        SAY 'ERROR CODE='VSAMCODE', MESSAGE='VSAMMSG    /*         */
        EXIT 8                          /*                        */
   END                                  /*                        */
                                        /*                        */
/* read the records from the file                                 */
                                        /*                        */
   DO NDX=1 TO 100                      /*                        */
      IF "RXVSAM"("READNEXT FILE("TESTFILE") INTO(RCD)")<>0 THEN LEAVE
      SAY RCD                           /*                        */
   END                                  /*                        */
                                        /*                        */
/* close and free the file                                        */
                                        /*                        */
   RC="RXVSAM"("CLOSE FILE("TESTFILE")")  /* close the file       */
                                        /*                        */
   RC="RXMVS"("FREE FILE("TESTFILE")")  /* free the file          */
                                        /*                        */
EXIT 0
```

Figure 81: Sample program using **RXMVS ALLOCATE**

```
THIS IS TEST RCD NUMBER=1
THIS IS TEST RCD NUMBER=2
THIS IS TEST RCD NUMBER=3
THIS IS TEST RCD NUMBER=4
THIS IS TEST RCD NUMBER=5
THIS IS TEST RCD NUMBER=6
THIS IS TEST RCD NUMBER=7
THIS IS TEST RCD NUMBER=8
THIS IS TEST RCD NUMBER=9
THIS IS TEST RCD NUMBER=10
THIS IS TEST RCD NUMBER=11
THIS IS TEST RCD NUMBER=12
THIS IS TEST RCD NUMBER=13
THIS IS TEST RCD NUMBER=14
THIS IS TEST RCD NUMBER=15
THIS IS TEST RCD NUMBER=16
THIS IS TEST RCD NUMBER=17
THIS IS TEST RCD NUMBER=18
THIS IS TEST RCD NUMBER=19
THIS IS TEST RCD NUMBER=20
THIS IS TEST RCD NUMBER=21
THIS IS TEST RCD NUMBER=22
THIS IS TEST RCD NUMBER=23
```

Figure 82: Output from **ALLOCATE** sample program

Deallocate a Data Set (FREE)

The **FREE** command is available for dynamic de-allocation of a data set.

Command Format

RXMVS FREE
FILE (filename)

Parameters

filename	Specify the valid file name (one to eight characters) used to reference the data set to be de-allocated. This is to be the same file name used in the **ALLOCATE** command.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from last operation.

`0`	Successful
`-3`	Syntax error
`12`	Unable to process statement.

RESULT Same as **MVSCODE** when invoked as external subroutine (**CALL "RXMVS"**).

MVSMSG Descriptive message.

Note: The following example is available for demonstration purposes. See the installation EXECS library member MAXXREXX1.

```
/*  REXX                                                          */
/*  DOC:  Sample to show file allocation/free                     */
/*                                                                */
/*  This sample MAX/REXX program uses the following features of the  */
/*  product:                                                      */
/*  RXMVS:   ALLOCATE, FREE                                       */
/*  RXVSAM:  OPEN, CLOSE, READNEXT, WRITE                         */
/*                                                                */
/*  To run this example do the following:                         */
/*  1.  set the DSN variable to a dataset name applicable to your    */
/*      environment.                                              */
/*  2.  set the TESTFILE variable to an appropriate ddname.       */
                                        /*                        */
   DSN='MXQUAL.TEST.FILE'               /* change this value !!!! */
   TESTFILE='SYSUT1'                    /* change this value !!!! */
                                        /*                        */
/* allocate the file as new,keep                                  */
                                        /*                        */
   IF "RXMVS"("ALLOCATE FILE("TESTFILE") DATASET("DSN")",
      "NEW CATLG SPACE(TRK) PRIMARY(00001) UNITNAME(SYSDA)",
      "DSORG(PS) RECFM(FB) BLKSIZE(06000) LRECL(00060)")<>0 THEN DO
        SAY 'UNABLE TO ALLOCATE'        /*                        */
        SAY 'Check the variable values set for DSN and TESTFILE'
        SAY 'ERROR CODE='MVSCODE', MESSAGE='MVSMSG    /*          */
        EXIT 8                          /*                        */
   END                                  /*                        */
                                        /*                        */
```

Figure 83: Sample program using **RXMVS FREE**

```
/* open the file for LOAD                                          */
                                        /*                         */

   IF "RXVSAM"("OPEN FILE("TESTFILE") LOAD")<>0 THEN DO
        SAY 'UNABLE TO OPEN FILE'       /*                         */
        SAY 'ERROR CODE='VSAMCODE', MESSAGE='VSAMMSG    /*          */
        EXIT 8                          /*                         */
   END                                  /*                         */
                                        /*                         */
/* write a series of records to the file                           */
                                        /*                         */
   DO NDX=1 TO 100                      /*                         */
      RCD="THIS IS TEST RCD NUMBER="NDX /*                         */
      RCD=SUBSTR(RCD,1,60,' ')          /*                         */
      IF "RXVSAM"("WRITE FILE("TESTFILE") FROM(RCD)")<>0 THEN LEAVE
   END                                  /*                         */
                                        /*                         */

/* close and free the file                                         */
                                        /*                         */
   RC="RXVSAM"("CLOSE FILE("TESTFILE")")  /* close the file        */
   RC="RXMVS"("FREE FILE("TESTFILE")")    /* free the file         */
                                        /*                         */
/* reallocate the file as shr - specify delete when finished       */
                                        /*                         */
   IF "RXMVS"("ALLOCATE FILE("TESTFILE") DATASET("DSN")",
              "SHR DELETE")<>0 THEN DO  /*                         */
        SAY 'UNABLE TO ALLOCATE'        /*                         */
        SAY 'ERROR CODE='MVSCODE', MESSAGE='MVSMSG    /*            */
        EXIT 8                          /*                         */
   END                                  /*                         */
                                        /*                         */
/* open the file for input - sequential                            */
                                        /*                         */
   IF "RXVSAM"("OPEN FILE("TESTFILE") SEQ")<>0 THEN DO
        SAY 'UNABLE TO OPEN FILE'       /*                         */
        SAY 'ERROR CODE='VSAMCODE', MESSAGE='VSAMMSG    /*          */
        EXIT 8                          /*                         */
   END                                  /*                         */
                                        /*                         */
/* read the records from the file                                  */
                                        /*                         */
   DO NDX=1 TO 100                      /*                         */
      IF "RXVSAM"("READNEXT FILE("TESTFILE") INTO(RCD)")<>0 THEN LEAVE
      SAY RCD                           /*                         */
   END                                  /*                         */
                                        /*                         */
/* close and free the file                                         */
                                        /*                         */
   RC="RXVSAM"("CLOSE FILE("TESTFILE")")  /* close the file        */
   RC="RXMVS"("FREE FILE("TESTFILE")")    /* free the file         */
                                        /*                         */
EXIT 0
```

Figure 83: Sample program using **RXMVS FREE**

```
THIS IS TEST RCD NUMBER=1
THIS IS TEST RCD NUMBER=2
THIS IS TEST RCD NUMBER=3
THIS IS TEST RCD NUMBER=4
THIS IS TEST RCD NUMBER=5
THIS IS TEST RCD NUMBER=6
THIS IS TEST RCD NUMBER=7
THIS IS TEST RCD NUMBER=8
THIS IS TEST RCD NUMBER=9
```

Figure 84: Sample Output from **FREE** program

Establish the RXMVS Host Command Environment (CONNECT)

The **CONNECT** command is available to establish RXMVS as a host command environment. This is not required to execute most of the other RXMVS commands. If required for a specific command, this will be noted with that command. In other situations, this command can be used for performance. If the REXX program issues several RXMVS commands, performance could be improved by first connecting to the environment. **CONNECT** may be invoked via the **CALL "RXMVS"** external routine, or as a function.

Command Format

RXMVS CONNECT

Parameters

No parameters for this command.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from last operation.

`0`	Successful
`-3`	Syntax error
`12`	Unable to process statement.

RESULT Same as **MVSCODE** when invoked as external subroutine (**CALL "RXMVS"**).

MVSMSG Descriptive message.

```
/*   REXX                                                        */
/*   CONNECT to the RXMVS host environment                       */

  RC=RXMVS("CONNECT")
  IF MVSCODE<>0 THEN DO
     SAY 'CONNECT ERROR RC='MVSCODE ' MVSMGS=' MVSMSG
        EXIT 8
  END
```

Figure 85: Connect to RXMVS Host Command Environment

Terminate the RXMVS Host Command (DISCONNECT)

The **DISCONNECT** command is available to terminate the RXMVS host command environment. This command must be issued prior to the termination of the REXX program if the **CONNECT** command has been issued earlier in the process.

Command Format

RXMVS DISCONN

Parameters

No parameters for this command.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE	Return code from last operation. `0` Successful `-3` Syntax error `12` Unable to process statement.
RESULT	Same as **MVSCODE** when invoked as external subroutine (**CALL "RXMVS"**).
MVSMSG	Descriptive message.

```
  RC=RXMVS("DISCONN")
  IF MVSCODE<>0 THEN DO
     SAY 'DISCONNECT ERROR RC='MVSCODE ' MVSMGS=' MVSMSG
        EXIT 8
  END
```

Figure 86: Disconnect from RXMVS Host Command Environment

Common Processing Routines

Common processing routines are not commands specific to MVS. Nevertheless, they are processes that are common to programs running in that environment. The commands in this section make these processes available to the REXX program using a single MAX/REXX statement.

Dynamically Assign a Value (ASSIGN)

The purpose of this statement is to provide a method of dynamically assigning a variable without the necessity of the **INTERPRET** function of REXX. In the standard REXX environment, the **INTERPRET** command cannot be compiled. To make use of a compiler for better performance, the **ASSIGN** command can replace the standard **INTERPRET** function when dynamically assigned variables are needed.

Command Format

```
RXMVS ASSIGN
          INTO(into_varname)
          FROM(from_varname)
```

Parameters

into_varname	This identifies a standard REXX variable into which the value is to be assigned.
from_varname	This identifies a standard REXX variable from which the value is to be extracted.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from last operation.

`0`	Successful
`-3`	Syntax error
`12`	Unable to process statement.

RESULT Same as **MVSCODE** when invoked as external subroutine (**CALL "RXMVS"**).

MVSMSG Descriptive message.

```
/* assign a variable to a dynamic field name                    */
/* without need for the INTERPRET function                      */

   DATA_FIELD="MARY JONES"
   FLDNAME="NAME_FIELD"
   RC=RXMVS("ASSIGN INTO("FLDNAME") FROM DATA_FIELD)")

/* result of this dynamic assignment:                           */
/* NAME_FIELD would have the value of "MARY JONES"              */
```

Figure 87: Assign a variable using **RXMVS ASSIGN**

Sort an Array of Variables (SORT)

Sort a variable array in either ascending or descending order. Specific positions can be used to determine the order, or the entire field can be included in the sort.

Command Format

RXMVS SORT
FROM(from_varname)
INTO(into_varname)
[START(strtnnn)]
[LENGTH(length)]
[SEQ(A|D)]

Parameters

from_varname	Identifies the name of a REXX variable that contains the data to be sorted. This variable must be a stem variable, and the `.0` level of the variable must contain the count of the number of variables in the array. The length of each stem element must be no more than 256 characters.
into_varname	Identifies the name of a REXX variable into which the output of the sort is to be assigned. Again, this must be a stem variable, and the `.0` level of the variable will be set to the count of variables in the array. This can be the same variable that is used in the **from_varname**, and the data will be assigned to the original variable names.

strtnnn	This is a numeric value of the starting position on which to sort the data. The value is relative to 1. The default is position 1. *This parameter is optional*.
length	This is a numeric value of the length of the field on which to sort the data. If this field is omitted, the length of the field will be considered from the starting position to the end of the field. Therefore, to sort on an entire field, omit both the start and length parameters.
A|D	This parameter indicates the order of the sort. Ascending sequence is default.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from last operation.

0	Successful
-3	Syntax error
4	The variable array contained no entries
12	Unable to process statement.

RESULT Same as **MVSCODE** when invoked as external subroutine (**CALL "RXMVS"**).

MVSMSG Descriptive message.

```
                                  /* read until end of file       */
DO UNTIL RC=8
    RC=RXVSAM("READNEXT FILE(FILE01) INTO(RCD)"
    REC#=REC#+1
    NAME_FROM_REC.REC#=SUBSTR(RCD,20,36)
END
                                  /* sort into name sequence      */
                                  /* sort entire field, ascending */
RC=RXMVS("SORT FROM(NAME_FROM_REC.) INTO(NAME_FROM_REC.)")
```

Figure 88: Sort a variable array

Date Processing Routines

The date processing function provides a frequently needed service in the programming environment. The date processing routines of MAX/REXX provide full century support. Services offered by these routines include conversion of dates from Julian to Gregorian and the reverse, and date calculation forward or backward, in both Julian and Gregorian format.

Calculate Date YYYY/MM/DD (CALCDATE)

The purpose of this statement is to provide a method to perform calculations, either forward or backward, on a Gregorian format date (yyyy/mm/dd). This command calculates a new Gregorian date by incrementing or reducing the date by the number of days requested.

Command Format

RXMVS CALCDATE
INTO(into_varname)
DATE(yyyy/mm/dd)
DAYS(daysnnn)

Parameters

into_varname	This identifies a standard REXX variable into which the value is to be assigned.
yyyy/mm/dd	Format of the Gregorian date.
daysnnn	Numeric value, assumed positive unless specified as `-n`, by which to adjust the date.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from last operation.
`0` Successful
`-3` Syntax error
`12` Unable to process statement.

RESULT Same as **MVSCODE** when invoked as external subroutine (**CALL "RXMVS"**).

MVSMSG Descriptive message.

into_varname_DAY Returns the day of the resulting date, such as `MONDAY`.

into_varname_MONTH Returns the month of the resulting date, such as `JANUARY`.

into_varname_DOW Returns the day of the week as a numeric value with `MONDAY=1`, etc.

```
/*   add 30 days grace period to due_date                          */

  DUE_DATE='2001/03/20'
  NUMBER_OF_DAYS='30'

  RC=RXMVS("CALCDATE DATE("DUE_DATE") DAYS("NUMBER_OF_DAYS")",
           "INTO(NEW_DATE_")

/*   convert the result to Julian                                  */

  RC=RXMVS(" DATE2JUL DATE("NEW_DATE") INTO(JULIAN_DATE)")

/*   show due date in Julian along with day of the week            */

  SAY "PAYMENT IS DUE:" JULIAN_DATE_DAY","JULIAN_DATE
```

Figure 89: Compute new Gregorian date value

Calculate Julian Date YYYY/DDD (CALCJUL)

The purpose of this statement is to provide a method to perform calculations, either forward or backward, to a Julian format date (yyyy/ddd). This command calculates a new Julian date by incrementing or reducing the date by the number of days requested.

Command Format

RXMVS CALCJUL
INTO(into_varname)
JDATE(yyyy/ddd)
DAYS(daysnnn)

Parameters

into_varname	This identifies a REXX variable into which the result of any of the commands will be returned. This is required for all of the commands.
yyyy/ddd	Format of the Julian date.
daysnnn	Numeric value, assumed positive unless specified as `-n`, by which to adjust the date.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE	Return code from last operation. `0` Successful `-3` Syntax error `12` Unable to process statement.
RESULT	Same as **MVSCODE** when invoked as external subroutine (**CALL "RXMVS"**).
MVSMSG	Descriptive message.
into_varname_DAY	Returns the day of the resulting date, such as `MONDAY`.
into_varname_MONTH	Returns the month of the resulting date, such as `JANUARY`.
into_varname_DOW	Returns the day of the week as a numeric value, with `MONDAY=1`, etc.

```
/*   add 30 days grace period to due_date                          */

  DUE_DATE='2001/101'
  NUMBER_OF_DAYS='30'

  RC=RXMVS("CALCDATE JDATE("DUE_DATE") DAYS("NUMBER_OF_DAYS")",
           "INTO(NEW_DATE_")

/*   convert the result to Gregorian                               */

  RC=RXMVS("JUL2DATE JDATE("NEW_DATE") INTO(GREG_DATE)")

/*   show due date in Gregorian along with day of the week         */

 SAY "PAYMENT IS DUE:" GREG_DATE_DAY","GREG_DATE
```

Figure 90: Compute new Julian date value

Convert Gregorian Date into Julian Format (DATE2JUL)

The purpose of this statement is to provide a method to convert a Gregorian format date (yyyy/mm/dd) into a Julian format (yyyy/ddd). This command converts the date provided in **DATE** from Gregorian to Julian.

Command Format

```
RXMVS DATE2JUL
          INTO(into_varname)
          DATE(yyyy/mm/dd)
```

Parameters

into_varname	This identifies a REXX variable into which the result of any of the commands will be returned.
yyyy/mm/dd	Format of the Gregorian date.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from last operation.

0	Successful
-3	Syntax error
12	Unable to process statement.

RESULT Same as **MVSCODE** when invoked as external subroutine (**CALL "RXMVS"**).

MVSMSG Descriptive message.

into_varname_DAY Returns the day of the resulting date, such as `MONDAY`.

into_varname_MONTH Returns the month of the resulting date, such as `JANUARY`.

into_varname_DOW Returns the day of the week as a numeric value, with `MONDAY=1`, etc.

```
/*   add 30 days grace period to due_date                          */

  DUE_DATE='2001/03/20'
  NUMBER_OF_DAYS='30'

  RC=RXMVS("CALCDATE DATE("DUE_DATE") DAYS("NUMBER_OF_DAYS")",
           "INTO(NEW_DATE_")

/*   convert the result to Julian                                  */

  RC=RXMVS("DATE2JUL DATE("NEW_DATE") INTO(JULIAN_DATE)")

/*   show due date in Julian along with day of the week            */

  SAY "PAYMENT IS DUE:" JULIAN_DATE_DAY","JULIAN_DATE
```

Figure 91: Convert Gregorian date to Julian

Convert Julian Date into Gregorian Format (JUL2DATE)

The purpose of this statement is to provide a method to convert a Julian format date (yyyy/ddd) into a Gregorian format (yyyy/mm/dd). This command converts the date provided in **JDATE** from Julian to Gregorian.

Command Format

RXMVS JUL2DATE
INTO(into_varname)
JDATE(yyyy/ddd)

Parameters

into_varname	This identifies a REXX variable into which the result of any of the commands will be returned. This is required for all of the commands.
yyyy/ddd	Format of the Julian date.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE	Return code from last operation. `0` Successful `-3` Syntax error `12` Unable to process statement.
RESULT	Same as **MVSCODE** when invoked as external subroutine (**CALL "RXMVS"**).
MVSMSG	Descriptive message.
into_varname_DAY	Returns the day of the resulting date, such as `MONDAY`.
into_varname_MONTH	Returns the month of the resulting date, such as `JANUARY`.
into_varname_DOW	Returns the day of the week as a numeric value, with `MONDAY=1`, etc.

```
/*   add 30 days grace period to due_date                          */

  DUE_DATE='2001/101'
  NUMBER_OF_DAYS='30'

  RC=RXMVS("CALCDATE JDATE("DUE_DATE") DAYS("NUMBER_OF_DAYS")",
           "INTO(NEW_DATE_")

/*   convert the result to Gregorian                               */

  RC=RXMVS("JUL2DATE JDATE("NEW_DATE") INTO(GREG_DATE)")

/*   show due date in Gregorian along with day of the week         */

  SAY "PAYMENT IS DUE:" GREG_DATE_DAY","GREG_DATE
```

Figure 92: Convert Julian date to Gregorian

Calculate Difference Between Two Julian Dates (CALCJDIF)

The purpose of the **CALCJDIF** command is to calculate the number of days between two Julian dates.

Command Format

RXMVS CALCJDIF
JDATE (yyyy/ddd)
JDATE2 (yyyy/ddd)
INTO (varname)

Parameters

yyyy/ddd	Format of the Julian date to pass.
varname	REXX variable name into which the result of the calculation will be returned. If the first date is greater than the second date, this value will be returned with a minus sign preceding the number.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from last operation.
`0` Successful
`-3` Syntax error
`12` Unable to process statement.

RESULT Same as **MVSCODE** when invoked as external subroutine (**CALL "RXMVS"**).

MVSMSG Descriptive message.

Note: The following example is available for demonstration purposes. See the installation EXECS library member MAXXREXX4.

Calculate Difference Between Two Gregorian Dates (CALCGDIF)

The purpose of the **CALCGDIF** command is to calculate the number of days between two Gregorian dates.

Command Format

RXMVS CALCGDIF
DATE (yyyy/dd/mm)
DATE2 (yyyy/dd/mm)
INTO (varname)

Parameters

yyyy/dd/mm	Format of the Gregorian date to pass.
varname	REXX variable name into which the result of the calculation will be returned. If the first date is greater than the second date, this value will be returned with a minus sign preceding the number.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from last operation.

`0`	Successful
`-3`	Syntax error
`12`	Unable to process statement.

RESULT Same as **MVSCODE** when invoked as external subroutine (**CALL "RXMVS"**).

MVSMSG Descriptive message.

Note: The following example is available for demonstration purposes. See the installation EXECS library member MAXXREXX4.

```
/*  REXX                                                          */
/*  DOC:  Sample to show various date conversions and calculations*/
/*                                                                */
/*  This sample MAX/REXX program uses the following features of the */
/*  product:                                                      */
/*  RXMVS:   DATE2JUL, CALCGDIF, CALCJDIF                         */
/*                                                                */
                                        /*                        */
   START_DATE='1998/12/01'              /* start date             */
   FINAL_DATE='2003/06/30'              /* end date               */
   DIFF=''                              /*                        */
                                        /*                        */
/* calculate the difference between two gregorian dates          */
                                        /*                        */
   CALL 'RXMVS' 'CALCGDIF DATE('START_DATE') DATE2('FINAL_DATE')',
                'INTO(DIFF)'
   IF MVSCODE<>0 THEN                   /*                        */
     CALL CLEANUP                       /*                        */
                                        /*                        */
   SAY 'DATE 1='START_DATE              /*                        */
   SAY 'DATE 2='FINAL_DATE              /*                        */
   SAY 'Difference between gregorian dates='DIFF   /*             */
                                        /*                        */
```

Figure 93: Sample program using RXMVS date functions

```
/* convert the dates to julian format                                  */
                                        /*                             */
   CALL 'RXMVS' 'DATE2JUL DATE('START_DATE') INTO('JUL_START')'
   IF MVSCODE<>0 THEN                   /*                             */
     CALL CLEANUP                       /*                             */
                                        /*                             */
   CALL 'RXMVS' 'DATE2JUL DATE('FINAL_DATE') INTO('JUL_FINAL')'
   IF MVSCODE<>0 THEN                   /*                             */
     CALL CLEANUP                       /*                             */
                                        /*                             */
/* show results from one of the date conversions                       */
                                        /*                             */
   SAY '                          '     /*                             */
   SAY 'Day of the week='JUL_FINAL_DAY    /* show some of the           */
   SAY 'Number of the day='JUL_FINAL_DOW  /* results returned           */
   SAY 'Month of the year='JUL_FINAL_MONTH  /* from conversion           */
   SAY 'Date returned  ='JUL_FINAL      /*                             */
                                        /*                             */

/* calculate the days difference between the two julian dates          */
/* --  reverse the order of the dates                                  */
                                        /*                             */
   CALL 'RXMVS' 'CALCJDIF JDATE('JUL_FINAL') JDATE2('JUL_START')',
                'INTO(DIFF)'
   IF MVSCODE<>0 THEN                   /*                             */
     CALL CLEANUP                       /*                             */
                                        /*                             */
   SAY '                          '     /*                             */
   SAY 'DATE 1='JUL_FINAL               /*                             */
   SAY 'DATE 2='JUL_START               /*                             */
   SAY 'Difference between julian dates='DIFF   /*                     */
                                        /*                             */
EXIT 0
CLEANUP:                                /*                             */
   SAY 'MVSCODE='MVSCODE                /* show error code             */
   SAY 'MVSMSG ='MVSMSG                 /* show error message          */
EXIT 8
```

Figure 93: Sample program using RXMVS date functions

```
DATE 1=1998/12/01
 DATE 2=2003/06/30
 Difference between gregorian dates=1672

 Day of the week=MONDAY
 Number of the day=1
 Month of the year=JUNE
 Date returned  =2003/181

 DATE 1=2003/181
 DATE 2=1998/335
 Difference between julian dates=-1672
```

Figure 94: Output from sample date program

Character Processing Routines

Although REXX has many built-in functions to process character strings, REXX has no way of handling characters that may be in different code pages. The character processing routines of MAX/REXX enhance REXX with functions to convert characters between different code pages and to process characters in specific code pages and character classes, including EBCDIC, ASCII and Unicode.

Convert Characters to another Code Page (CONVERT)

The **CONVERT** command converts a character string encoded in one code page to a character string encoded in a different code page. Optionally, line-end characters in the character string can be replaced with a different sequence of line-end characters. Any characters in the string that do not have equivalent code points in the target code page will be replaced with the substitution character of the target code page (`X'3F'` for EBCDIC, `X'1A'` for ASCII, or `X'FFFD'` for Unicode).

Command Format

```
RXMVS CONVERT
          FROM(from_varname)
          INTO(into_varname)
          FROMCODEPAGE(from_codepage)
          TOCODEPAGE(to_codepage)
          [CR|LF|NL|CRLF|LFCR|CRNL]
```

Parameters

from_varname	Identifies the name of a REXX variable that contains the string of characters to be converted.
into_varname	Identifies the name of a REXX variable to which the converted character string will be assigned.
from_codepage	Identifies the number of the code page in which the **from_varname** characters are encoded. **FROMCODEPAGE** may be specified as **FROMCP**.
to_codepage	Identifies the code page number into which the characters are to be encoded. **TOCODEPAGE** may be specified as **TOCP**.
CR | LF | NL | CRLF | LFCR | CRNL	Indicates that any end-of-line character(s) in the **from_varname** are to be replaced with the specified end-of-line character(s) of the **to_codepage**. **CR** specifies the carriage return character. **LF** specifies the line feed character. **NL** specifies the new line character. **CRLF**, **LFCR** and **CRNL** specify the two-character combinations. Not all ASCII code pages define the new line character and, when converting to these code pages, **LF** will be used if **NL** is specified. When converting from Unicode, the line separator character `X'2028'` is recognized as an end-of-line indicator in addition to **CR**, **LF** and **NL**. If none of these options are specified, end-of-line characters will be directly converted to their equivalent in the new code page.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from the last operation.

0	Successful
-3	Syntax or environment error
8	Invalid character; into_varname will be set to the zero-based offset of the first invalid character
12	Code page not supported

RESULT Same as **MVSCODE** when invoked as an external subroutine (**CALL "RXMVS"**).

MVSMSG Descriptive message.

Determine if a Code Page is EBCDIC, ASCII or Unicode (CPCLASS)

The **CPCLASS** command can be used to determine if a specific code page is EBCDIC, ASCII or Unicode.

Command Format

```
RXMVS CPCLASS
        CODEPAGE(codepage)
        INTO(into_varname)
```

Parameters

codepage	Identifies the number of the code page for which the class is to be determined.
into_varname	Identifies the name of a REXX variable to which the code page's class will be assigned. The assigned value will be one of the following: EBCDIC, ASCII, or Unicode.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from the last operation.

0	Successful
-3	Syntax or environment error
12	Code page not supported

RESULT Same as **MVSCODE** when invoked as an external subroutine (**CALL "RXMVS"**).

MVSMSG Descriptive message.

List the Supported Code Pages (CPLIST)

The **CPLIST** command can be used obtain a list of the code pages supported by MAX/REXX.

Command Format

```
RXMVS CPLIST
        INTO(into_varname)
```

Parameters

into_varname	Identifies the name of a REXX stem variable to which the list of supported code pages will be assigned. The name should normally end in a period. The name with a '0' appended will contain the number of code pages in the list. The names with appended numbers starting with '1' will each contain a code page number. The code pages will be listed in ascending order.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE	Return code from the last operation. 0 Successful -3 Syntax or environment error
RESULT	Same as **MVSCODE** when invoked as an external subroutine (**CALL "RXMVS"**).
MVSMSG	Descriptive message.

Return a Code Page Description (CPNAME)

The **CPNAME** command can be used obtain a description of a specific code page.

Command Format

```
RXMVS CPNAME
        CODEPAGE(codepage)
        INTO(into_varname)
```

Parameters

codepage	Identifies the number of the code page for which the class is to be determined.
into_varname	Identifies the name of a REXX variable to which a descriptive string will be assigned.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from the last operation.

`0`	Successful
`-3`	Syntax or environment error
`12`	Code page not supported

RESULT Same as **MVSCODE** when invoked as an external subroutine (**CALL "RXMVS"**).

MVSMSG Descriptive message.

Encode to Base64 (ENCODE64)

The **ENCODE64** command encodes data using Base64, allowing arbitrary binary data to be transmitted as text. Base64 is often used in e-mail attachments and XML documents.

Command Format

```
RXMVS ENCODE64
        FROM(from_varname)
        INTO(to_varname)
        [ EBCDIC ]
```

Parameters

from_varname	Identifies the name of a REXX variable that contains the string of characters to be converted.
into_varname	Identifies the name of a REXX variable to which the encoded character string will be assigned.
EBCDIC	Indicates that the Base64 characters are to be generated as EBCDIC text. The default is to generate ASCII/UTF-8 text.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE	Return code from the last operation. 0 Successful -3 Syntax or environment error
RESULT	Same as **XMLCODE** when invoked as an external subroutine (**CALL "RXMVS"**).
MVSMSG	Descriptive message.

Decode from Base64 (DECODE64)

The **DECODE64** command decodes data that was previously encoded using Base64. Base64 is often used in e-mail attachments and XML documents.

Command Format

```
RXMVS DECODE64
          FROM(from_varname)
          INTO(to_varname)
          [ EBCDIC ]
```

Parameters

from_varname	Identifies the name of a REXX variable that contains the string of Base64 characters to be converted.
into_varname	Identifies the name of a REXX variable to which the decoded character string will be assigned.
EBCDIC	Indicates that the Base64 characters are EBCDIC text. The default is ASCII/UTF-8 text.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE	Return code from the last operation. 0 Successful -3 Syntax or environment error
RESULT	Same as **XMLCODE** when invoked as an external subroutine (**CALL "RXMVS"**).
MVSMSG	Descriptive message.

Test Characters

These commands test each character in a string encoded in a specific code page to determine if they belong to a specified set. A string containing the results of each test is returned. The result string contains as many EBCDIC '1' or '0' characters as there are characters in the original character string. A '1' indicates that the character belongs to the specified set and a '0' indicates that the character does not.

The sets are similar to the "C" language functions of the same name, but are defined for each character in a code page using the Unicode character class definitions. Following is a list of the sets:

Sets	
ISALNUM	Alphanumeric characters.
ISALPHA	Alphabetic characters.
ISBLANK	Blanks and horizontal tab characters.
ISCNTRL	Control characters.
ISDIGIT	Digit characters.
ISGRAPH	Non-blank printing characters.
ISLOWER	Lowercase letters.
ISPRINT	Printing characters.
ISPUNCT	Punctuation characters.
ISSPACE	Spacing characters, including blanks, tabs, carriage return, line feed and new line characters.
ISUPPER	Uppercase letters.
ISXDIGIT	Hexadecimal digits (0-9, A-Z, a-z).

Command Format

```
RXMVS     { ISALNUM | ISALPHA | ISBLANK | ISCNTRL |
            ISDIGIT | ISGRAPH | ISLOWER | ISPRINT |
            ISPUNCT | ISSPACE | ISUPPER | ISXDIGIT }
                  FROM(from_varname)
                  INTO(into_varname)
                  CODEPAGE(codepage)]
```

Parameters

from_varname	Identifies the name of a REXX variable that contains the string of characters to be tested.
into_varname	Identifies the name of a REXX variable to which the test results will be assigned.
codepage	Identifies the number of the code page in which the characters are encoded.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from the last operation.

`0`	Successful
`-3`	Syntax or environment error
`8`	Code page is not for fixed-width characters
`12`	Code page not supported

RESULT Same as **MVSCODE** when invoked as an external subroutine (**CALL "RXMVS"**).

MVSMSG Descriptive message.

```
STRING="A1B2C#"
RC=RXMVS("ISALPHA FROM(STRING) INTO(TEST) CODEPAGE(37)")
/* TEST should contain "101010" */
```

Figure 95: Test if characters are alphabetic

Convert Characters to Lowercase (TOLOWER)

The **TOLOWER** command converts any uppercase characters in a string to their lowercase equivalents in a specific code page. For example, 'Ä' will be converted to 'ä'. Any characters that are not uppercase are left unchanged. The converted string will have exactly as many characters as the original string.

Command Format

```
RXMVS TOLOWER
        FROM(from_varname)
        INTO(into_varname)
        CODEPAGE(codepage)]
```

Parameters

from_varname	Identifies the name of a REXX variable that contains the string of characters to be converted.
into_varname	Identifies the name of a REXX variable to which the converted character string will be assigned.
codepage	Identifies the number of the code page in which the characters are encoded.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from the last operation.

`0`	Successful
`-3`	Syntax or environment error
`12`	Code page not supported

RESULT Same as **MVSCODE** when invoked as an external subroutine (**CALL "RXMVS"**).

MVSMSG Descriptive message.

Convert Characters to Uppercase (TOUPPER)

The **TOUPPER** command converts any lowercase characters in a string to their uppercase equivalents in a specific code page. For example, 'ä' will be converted to 'Ä'. Any characters that are not lowercase are left unchanged. The converted string will have exactly as many characters as the original string. Lowercase characters such as ß which have an uppercase equivalent of more than one character will be left unchanged.

Command Format

```
RXMVS TOUPPER
          FROM(from_varname)
          INTO(into_varname)
          CODEPAGE(codepage)]
```

Parameters

from_varname	Identifies the name of a REXX variable that contains the string of characters to be converted.
into_varname	Identifies the name of a REXX variable to which the converted character string will be assigned.
codepage	Identifies the number of the code page in which the characters are encoded.

Special Variables

Upon completion, RXMVS will set the following REXX variables to new values.

MVSCODE Return code from the last operation.

`0`	Successful
`-3`	Syntax or environment error
`12`	Code page not supported

RESULT Same as **MVSCODE** when invoked as an external subroutine (**CALL "RXMVS"**).

MVSMSG Descriptive message.

CHAPTER 6: TEXT FILE PROCESSING

Summary

The text processing feature of MAX/REXX provides an easy way to read text data into a REXX program. The feature provides special processing for text data containing XML documents, Comma Separated Variable (CSV) data, and Tab Delimited (TAB) data.

Text Processing Commands	
OPEN	Begin processing text data.
CLOSE	End processing text data.
REWIND	Restart processing text data.
SKIP	Skip forward into text data.
READNEXT	Read text data.
SCAN	Summarize text data.

Special Variables

TEXTCODE

The special variable **TEXTCODE** will always contain the return code following any invocation of RXTEXT. This variable is always available whether RXTEXT was invoked via the host command environment (**ADDRESS RXTEXT**), an external routine (**CALL "RXTEXT"**), or as a function (**RXTEXT(...)**). This variable may be used to test for the successful completion of a RXTEXT statement regardless of how RXTEXT was invoked. This facilitates the use of a common error routine to handle all RXTEXT errors.

TEXTMSG

The special variable **TEXTMSG** will always contain a descriptive message following any invocation of RXTEXT. This variable is always available whether RXTEXT was invoked via the host command environment (**ADDRESS RXTEXT**), an external routine (**CALL "RXTEXT"**), or as a function (**RXTEXT(...)**). This variable contains descriptive text that can be used to further assist in determining the problem.

TEXTTRACE

When the special variable **TEXTTRACE** is assigned to the value '`TRACE`', RXTEXT will write the resolved statement to SYSTSPRT. This special trace uses the REXX **SAY** function and may be used in any REXX environment. Tracing can be stopped by assigning **TEXTTRACE** to any value other than '`TRACE`'. This tracing will occur whether RXTEXT was invoked via the host command environment (**ADDRESS RXTEXT**), an external routine (**CALL "RXTEXT"**), or as a function (**RXTEXT(...)**).

Text File Processing Statements

Begin Processing (OPEN)

The **OPEN** command is used to begin processing text data. **OPEN** may only be invoked via the **CALL "RXTEXT"** external function or the **RXTEXT( )** function call.

Command Format

```
RXTEXT OPEN
        FILE({ddname | file_handle})
        [ FROM(variable_name) ]
        [ FILECP(file_codepage) ]
        [ REXXCP(rexx_codepage) ]
        [ FORMAT(TEXT|XML|CSV|TAB) ]
```

Parameters

ddname	Specifies the name of a **DD** to which is allocated a sequential data set or PDS member from which the data will be read. For a data set with fixed-length records that is not opened with **FORMAT(XML)**, trailing banks will be stripped from each record. If the fixed-length record is entirely blank, it will be processed as if it contained a single blank character.
file_handle	Specifies the 1 to 8 digit decimal number of a UNIX Systems Services file handle which is open for reading and from which the data will be read.
variable_name	Specifies that the data is actually to be read from a REXX variable, not the named **DD** or file handle. The **FILE** parameter is still required to associate the related commands with the variable.
file_codepage	Specifies a code page number from 1 to 65535, which will be used if the file does not contain a Unicode Byte Order mark, the XML document does not have a valid encoding directive or the UNIX files is not tagged with a recognized CCSID. The code page number must be the same class (ASCII or EBCDIC) as the file contents. If not specified, the file code page will be assumed to be `1047` for an EBCDIC file or `819` (ISO-8859-1) for an ASCII file.
rexx_codepage	Specifies a code page number from 1 to 65535, into which the data values will be translated. If not specified for a XML document, the data values will be returned in UTF-8. If not specified for a non-XML file, the data values will be returned in the file code page.
TEXT | XML | CSV | TAB	Specifies that the file contains a XML document, Comma Separated Variable data (CSV) or Tab Delimited data (TAB). If not specified, the default is `TEXT` and no special processing of the data is performed.

Special Variables

Upon completion, RXTEXT will set the following REXX variables to new values.

TEXTCODE Return code from last operation.
- `0` Successful
- `-3` Syntax or environment error
- `12` Unable to process statement

RESULT Same as **TEXTCODE** when invoked as an external subroutine (**CALL "RXTEXT"**).

TEXTMSG Descriptive message.

End Processing (CLOSE)

The **CLOSE** command ends processing of the text data that begun with **OPEN**. Once **CLOSE** has been performed for all **OPEN** commands, the RXTEXT host command environment will be terminated.

Command Format

```
RXTEXT CLOSE
       FILE({ddname | file_handle})
```

Parameters

ddname	Specifies the name of a **DD** which was referenced in a previously executed RXTEXT **OPEN** statement.
file_handle	Specifies the number of a UNIX Systems Services file handle which was referenced in a previously executed RXTEXT **OPEN** statement.

Special Variables

Upon completion, RXTEXT will set the following REXX variables to new values.

TEXTCODE Return code from last operation.
- `0` Successful
- `-3` Syntax or environment error
- `12` Unable to process statement

RESULT Same as **TEXTCODE** when invoked as an external subroutine (**CALL "RXTEXT"**).

TEXTMSG Descriptive message.

Restart Processing (REWIND)

The **REWIND** command restarts processing at the beginning of the text data. The effect is the same as a **CLOSE** followed by an **OPEN**, but is much faster.

Command Format

```
RXTEXT REWIND
       FILE({ddname | file_handle})
```

Parameters

ddname	Specifies the name of a **DD** which was referenced in a previously executed RXTEXT **OPEN** statement.
file_handle	Specifies the number of a UNIX Systems Services file handle which was referenced in a previously executed RXTEXT **OPEN** statement.

Special Variables

Upon completion, RXTEXT will set the following REXX variables to new values.

TEXTCODE Return code from last operation.

`0`	Successful
`-3`	Syntax or environment error
`12`	Unable to process statement

RESULT Same as **TEXTCODE** when invoked as an external subroutine (**CALL "RXTEXT"**).

TEXTMSG Descriptive message.

Skip Forward (SKIP)

The **SKIP** command reads the one or more text lines or second-level XML tags without returning their contents. The combination of **REWIND** and **SKIP** can be used to reposition to any line or second-level tag.

Command Format

```
RXTEXT SKIP
       FILE({ddname | file_handle})
       COUNT(count)
```

Parameters

ddname	Specifies the name of a **DD** which was referenced in a previously executed RXTEXT **OPEN** statement.
file_handle	Specifies the number of a UNIX Systems Services file handle which was referenced in a previously executed RXTEXT **OPEN** statement.
count	Specifies the number of text lines or second-level XML tags to be skipped. The number must be from 1 to 999999999.

Special Variables

Upon completion, RXTEXT will set the following REXX variables to new values.

TEXTCODE Return code from last operation.

`0`	Successful
`-3`	Syntax or environment error
`8`	End of file
`12`	Unable to process statement

RESULT Same as **TEXTCODE** when invoked as an external subroutine (**CALL "RXTEXT"**).

TEXTMSG Descriptive message.

Read Data (READNEXT)

The **READNEXT** command reads one or more text lines or second-level XML tags and returns the data read into REXX variables.

Command Format

```
RXTEXT READNEXT
       FILE({ddname | file_handle})
       [ INTO(variable_prefix) ]
       [ COUNT(count) ]
```

Parameters

ddname	Specifies the name of a **DD** which was referenced in a previously executed RXTEXT **OPEN** statement.
file_handle	Specifies the number of a UNIX Systems Services file handle which was referenced in a previously executed RXTEXT **OPEN** statement.
variable_prefix	Specifies the prefix of the variables that will be set with the contents of the lines or tags read. These variables are described below. If the **INTO** parameter is not specified for an XML document, the tag names will be used as the variable names, as described below.
count	Specifies the maximum number of text lines or second-level XML tags that are to be read. The number must be from 1 to 999999999. If not specified, the default is `1`. If at least one line or tag is read, RXTEXT will not set **TEXTCODE** to `8` when less than the specified number of lines or tags remain in the file. If the **INTO** parameter is not specified for an XML document, the **COUNT** parameter must also be omitted.

Special Variables

Upon completion, RXTEXT will set the following REXX variables to new values.

TEXTCODE Return code from last operation.

`0`	Successful
`-3`	Syntax or environment error
`8`	End of file with no remaining data
`12`	Unable to process statement

RESULT Same as **TEXTCODE** when invoked as an external subroutine (**CALL "RXTEXT"**).

TEXTMSG Descriptive message.

Upon successful completion of a **READNEXT** command without an **INTO** parameter for an XML document, RXTEXT will set REXX variables to the tag values using the tag names as variable names. The document root tag will be ignored. The tag names will be converted to uppercase using code page `1047` and any hyphen characters converted to underscores and colon characters converted to periods. **TEXTCODE** will be set to `12` if, after conversion, the tag name is not a valid REXX variable name.

Upon successful completion of a **READNEXT** command with an **INTO** parameter for an XML document, RXTEXT will set the following REXX stem variables to new values. In each case, the "`.0`" variable of the stem will be set to the number of items in the stem. Also in each case, the "`.1`" variable of the stem will be set to the root tag information.

prefixTEXT.n	For an attribute or a tag node, the attribute or tag name, converted to EBCDIC code page `1047`. For a text node, the text, converted to the REXX code page as determined by **OPEN**.
prefixFIRST.n	Item number of the first subordinate node. Will be `0` if there are no subordinate nodes.
prefixLAST.n	Item number of the last subordinate node. Will be `0` if there are no subordinate nodes.
prefixTYPE.n	`0` if the node is text, `1` if the node is an attribute, `2` if the node is a tag.

Upon successful completion, RXTEXT will set the following REXX stem variables to new values for a CSV or TAB file.

prefixTEXT.0	Number of lines read.
prefixTEXT.n.0	Number of data items in each line.
prefixTEXT.n.n	Value of each data item.

Upon successful completion, RXTEXT will set the following REXX stem variables to new values for a text file that was not opened as XML, CSV or TAB.

prefixTEXT.0	Number of lines read.
prefixTEXT.n	Contents of each text line.

Example

Given the following document:

```
<?xml version="1.1">
<root>
  <customer number="1">
    <name>George</name>
    <address>Denver</address>
    <address>Colorado</address>
  </customer>
  <customer number="2">
    <name>Mike</name>
    <last>Smith</last>
    <address>Lakewood</address>
  </customer>
  <business>
    <name>MAX</name>
    <address>Denver</address>
    <address>Colorado</address>
  </business>
</root>
```

The result of RXTEXT **READNEXT FILE(file)** is:

Variable Name	Variable Value
CUSTOMER	
NUMBER	1
NAME	George
ADDRESS	Colorado
TEXTCODE	4

The result of RXTEXT **READNEXT FILE(file) INTO(XML_) COUNT(3)** is:

Item (n)	XML_TEXT.n	XML_FIRST.n	XML_LAST.n	XML_TYPE.n
1	root	2	4	2
2	customer	5	8	2
3	customer	9	12	2
4	business	13	15	2
5	number	16	16	1
6	name	17	17	2
7	address	18	18	2
8	address	19	19	2
9	number	20	20	1
10	name	21	21	2
11	last	22	22	2
12	address	23	23	2
13	name	24	24	2
14	address	25	25	2
15	address	26	26	2
16	1	0	0	0
17	George	0	0	0
18	Denver	0	0	0
19	Colorado	0	0	0
20	2	0	0	0
21	Mike	0	0	0
22	Smith	0	0	0
23	Lakewood	0	0	0
24	MAX	0	0	0
25	Denver	0	0	0
26	Colorado	0	0	0

Scan Data (Scan)

The **SCAN** command reads one or more text lines or second-level XML tags and returns a summary of the data read into REXX variables. The **SCAN** command can be used to determine the structure of a XML document or the number of data items on each CSV or TAB line.

Command Format

```
RXTEXT SCAN
        FILE({ddname | file_handle})
        [ INTO(variable_prefix) ]
        [ COUNT(count) ]
```

Parameters

ddname	Specifies the name of a **DD** which was referenced in a previously executed RXTEXT **OPEN** statement.
file_handle	Specifies the number of a UNIX Systems Services file handle which was referenced in a previously executed RXTEXT **OPEN** statement.
variable_prefix	Specifies the prefix of the variables that will be set with the contents of the lines or tags read. These variables are described below.
count	Specifies the maximum number of text lines or second-level XML tags that are to be read. The number must be from 1 to 999999999. If not specified, the default is `1`. If at least one line or tag is read, RXTEXT will not set **TEXTCODE** to `8` when less than the specified number of lines or tags remain in the file.

Special Variables

Upon completion, RXTEXT will set the following REXX variables to new values.

TEXTCODE Return code from last operation.

`0`	Successful
`-3`	Syntax or environment error
`8`	End of file with no remaining data
`12`	Unable to process statement

RESULT Same as **TEXTCODE** when invoked as an external subroutine (**CALL "RXTEXT"**).

TEXTMSG Descriptive message.

Upon successful completion, RXTEXT will set the following REXX stem variables to new values for a XML document. In each case, the "`.0`" variable of the stem will be set to the number of items in the stem. Also in each case, the "`.1`" variable of the stem will be set to the root tag information.

prefixTEXT.n	For an attribute or a tag node, the attribute or tag name, converted to EBCDIC code page `1047`. For a text node, the first non-null text, converted to the REXX code page as determined by **OPEN**.
prefixFIRST.n	Item number of the first subordinate node. Will be `0` if there are no subordinate nodes.
prefixLAST.n	Item number of the last subordinate node. Will be `0` if there are no subordinate nodes.
prefixTYPE.n	`0` if the node is text, `1` if the node is an attribute, `2` if the node is a tag.
prefixMIN.n	Minimum number of occurrences of this tag.
prefixMAX.n	Maximum number of occurrences of this tag.

Upon successful completion, RXTEXT will set the following REXX variables to new values for a CSV or TAB file.

prefixCOUNT	Number of lines read.
prefixMIN	Minimum number of data items in each line.
prefixTEXT.0	Maximum number of data items in each line.
prefixTEXT.n	First non-null item value.

Upon successful completion, RXTEXT will set the following REXX variables to new values for a text file that was not opened as XML, CSV or TAB.

prefixCOUNT	Number of lines read.
prefixMIN	Minimum line length.
prefixMAX	Maximum line length.

Example

Given the following document:

```
<?xml version="1.1">
<root>
  <customer number="1">
    <name>George</name>
    <address>Denver</address>
    <address>Colorado</address>
  </customer>
  <customer number="2">
    <name>Mike</name>
    <last>Smith</last>
    <address>Lakewood</address>
  </customer>
  <business>
    <name>MAX</name>
    <address>Denver</address>
    <address>Colorado</address>
  </business>
</root>
```

The result of RXTEXT **SCAN FILE(file) INTO(XML_) COUNT(3)** is:

Item (n)	XML_TEXT.n	XML_FIRST.n	XML_LAST.n	XML_TYPE.n	XML_MIN.n	XML_MAX.n
1	root	2	3	2	1	1
2	customer	4	7	2	2	2
3	business	8	9	2	1	1
4	number	10	10	1	1	1
5	name	11	11	2	1	1
6	address	12	12	2	1	2
7	last	13	13	2	0	1
8	name	14	14	2	1	1
9	address	15	15	2	1	1
10	1	0	0	0	1	1
11	George	0	0	0	1	1
12	Denver	0	0	0	1	1
13	Smith	0	0	0	1	1
14	MAX	0	0	0	1	1
15	Denver	0	0	0	1	1

CHAPTER 7: INTERPRETIVE COMPILER FEATURE

Summary

Compile a REXX program into a object module ready for the linkage editor.

Syntax

```
RXCMPL MEMBER=NAME
       [,ENCRYPT | NOENCRYPT    ]
       [,COMP | NOCOMP          ]
```

Operands

MEMBER=NAME	**CSECT** name to be given to module.
ENCRYPT	**ENCRYPT** the output module.
NOENCRYPT	Do not **ENCRYPT** the output module.
COMP	Compress the module into its smallest size. This may alter the line numbering of program from its original line numbering.
NOCOMP	Preserve the original line numbers from the source program.

Allocations

The following **DDNAMES** must be allocated before invoking "I-COMPILER."

SOURCE	REXX source program.
SYSLIB	To SYS1.MACLIB.
SYSIN	Temporary DASD work area (LRECL=80).
SYSUT1	Temporary DASD work area.
SYSUT2	Temporary DASD work area.
SYSUT3	Temporary DASD work area.
SYSLIN	Output OBJECT module.
SYSTERM	Compile error messages.
SYSTSPRT	Output status report.

Sample JCL

This sample JCL may be found in member RXCMPLJ1 in the supplied JCL library.

```
//RXCMPL1  JOB 0,'COMPILE REXX PROGRAM',
//         CLASS=A,MSGCLASS=H
//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
//* DOC: Use MAX I-COMPILE a REXX pgm into a load module
//*      You should expect a return code 0 from RXCMPL1
//*      You should expect a return code 0 from LKED2
//*      You should expect a return code 0 from TEST3
//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
//* CHANGES:
//*      1) Set the value of MXQUAL to the high level qualifier for
//*         the MAX libraries
//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
//RXCMPL   PROC MXQUAL=MXS,               ** HIGH LEVEL QUALIFIER **
//              MXCL='*',                 ** SYSOUT CLASS         **
//              MEM='HELPRCMP'            ** REXX SOURCE PROGRAM  **
//*
//* STEP: RXCMPL1, compile a rexx source program into an object module
//*
//RXCMPL1  EXEC PGM=RXCMPL,REGION=8000K,
//              PARM='MEMBER=&MEM,COMP,ENCRYPT,ASM=ASMA90'
//STEPLIB  DD  DISP=SHR,DSN=&MXQUAL..MXRXV240.LOADLIB
//SYSLIB   DD  DISP=SHR,DSN=SYS1.MACLIB
//SYSPRINT DD  SYSOUT=&MXCL
//SYSTSPRT DD  SYSOUT=&MXCL
//SYSTSIN  DD  DUMMY
//SYSTERM  DD  SYSOUT=&MXCL
//SYSIEXEC DD  SYSOUT=&MXCL
//SYSCEXEC DD  DUMMY
//SOURCE   DD  DISP=SHR,DSN=&MXQUAL..MXRXV240.EXECS(&MEM)
//SYSLIN   DD  DSN=&&OBJECT,UNIT=VIO,SPACE=(80,(200,100)),
//              DISP=(MOD,PASS)
//SYSPUNCH DD  DSN=&&PUNCH,UNIT=VIO,SPACE=(80,(200,100)),
//              DISP=(MOD,PASS)
//INCLUDE  DD  DSN=&MXQUAL..MXRXV240.EXECS,DISP=SHR
//SYSIN    DD  DSN=&&SYSIN(&MEM),UNIT=VIO,SPACE=(1700,(3000,50,2)),
//              DCB=(LRECL=80,BLKSIZE=4096,RECFM=FB,DSORG=PO)
//SYSUT1   DD  DSN=&&SYSUT1,UNIT=VIO,SPACE=(1700,(600,100))
//SYSUT2   DD  DSN=&&SYSUT2,UNIT=VIO,SPACE=(1700,(300,50))
//SYSUT3   DD  DSN=&&SYSUT3,UNIT=VIO,SPACE=(1700,(300,50))
//*
```

Figure 96: Sample JCL

```
//* STEP: LKED2, link the object module generated in step RXCMPL1
//*
//LKED2   EXEC PGM=IEWL,COND=(0,NE,RXCMPL1),
//             PARM=(LET,LIST,REUS,RENT)
//SYSPRINT  DD SYSOUT=&MXCL
//SYSLIB    DD DSN=&MXQUAL..MXRXV240.LOADLIB,DISP=SHR
//SYSLMOD   DD DISP=SHR,DSN=&MXQUAL..MXRXV240.LOADLIB(&MEM)
//SYSLIN    DD DSN=&&OBJECT,DISP=(OLD,DELETE)
//SYSUT1   DD  DSN=&&SYSUT1,UNIT=VIO,SPACE=(600,(600,100))
//*
//* STEP: TEST3, execute the load module linked in step LKED2
//*
//TEST3   EXEC PGM=&MEM,COND=(0,NE,LKED2)
//STEPLIB DD   DISP=SHR,DSN=&MXQUAL..MXRXV240.LOADLIB
//SYSTSPRT  DD SYSOUT=&MXCL
//SYSTSIN   DD DUMMY
//LINKEDIT PEND
```

Figure 96: Sample JCL

ISPF Considerations

Special considerations when executing a compiled REXX program in ISPF.

Use the "**LANG (CREX)**" operand to execute a compiled REXX program that invokes ISPF dialog service (**ADDRESS ISPEXEC**) and references ISPF dialog variables as shown below.

ISPEXEC SELECT CMD (rexx pgm name) . . . LANG (CREX) . . .

or,

ISPSTART CMD (rexx pgm name) . . . LANG(CREX) . . .

When a REXX program does not share variables with any ISPF services such as Panels, Messages, Tables, File Tailoring, etc… then, there is no requirement to use **LANG(CREX)** operand.

APPENDIX A: RXVSAM RETURN CODES

Return Code	Description	Further Information
0	Successful Completion.	
-3	Syntax Error.	See special variable VSAMMSG.
4	Copybook match not found.	Valid with **SELECTCB** and **READ(all formats)** of files opened with **MAPDD**.
8	End of file reached. **READNEXT**, **READPREV**, **STARTBR**. Unable to process statement	
12	Possible VSAM error. See VSAM reference for Return Codes.	See special variable VSAMMSG.

APPENDIX B: RXSQL RETURN CODES

Return Code	Description	Further Information
0	Successful Completion.	
-3	Syntax Error.	See special variable SQLMSG.
-7 through -927	Error SQLCODES. See DB2 Messages and Codes.	See special variable SQLMSG.
8	Unable to process statement. Example: call attach connect failure, etc.	See special variable SQLMSG.
12	Unable to resolve statement. Example: unable to resolve variable, **CURSOR** not defined, **STATEMENT NAME** not defined, etc.	See special variable SQLMSG.
100 through 802	Warning SQLCODES. See DB2 Messages and codes.	See special variable SQLMSG.

APPENDIX C: RXMVS RETURN CODES

Return Code	Description	Further Information
0	Successful Completion.	
-3	Syntax Error.	See special variable MVSMSG.
4	**SORT**: no variables in array; 0 level =0. **ENQ**: resource not immediately available. **DIR**: no members were found in directory member statements; member already exists.	See special variable MVSMSG.
8	**ENQ**: task already has control of resource. **DEQ**: resource has already been released member statements: member not found.	See special variable MVSMSG.
12	Unable to resolve statement. Several specific errors – see MVSMSG for detail.	See special variable MVSMSG.
14	**ENQ**: task made previous request for control, does not have control	See special variable MVSMSG.
18	**ENQ**: limit of number of concurrent resource requests has been reached; task does not have control.	See special variable MVSMSG.

INDEX

R

S

T

U

V

W

X