BROADCOM®
MAINFRAME SOFTWARE

RDX REXX Db2 eXtentions 20.0

Table of Contents

Announcements and News

New Training Course

A new web-based training course is now available from Mainframe Software Education: **Smart/RESTART: Restarting Db2 Applications and Resolving Application Performance Issues 200 (06RAI20040)**
This 30-minute course explains how to:

- Restart Db2 applications with or without code changes
- Restart Db2 applications after making changes to resolve an application performance issue

Training Access

Web-based training is available at no charge to customers who are on active maintenance. For information about accessing Broadcom web-based training, see Mainframe Software Education and also view our course catalog. If you would like to share your ideas for Broadcom mainframe product training, subscribe to the Mainframe Education Community, where we share calls for input.

Mainframe Community:

Learn, connect, and share knowledge and experiences with other users by joining the Broadcom Mainframe Software Community.

Security Advisories Consolidated CSV File

Broadcom offers a .CSV file that contains a consolidated list of security advisories affecting all supported Broadcom mainframe products. This file lets you easily search the Common Vulnerabilities and Exposures (CVE) information. You can also access the Security Advisory articles that include more details and context about the security or integrity exposure. Broadcom updates this file daily. For CSV download instructions, see this Broadcom Support article (login required).

Mainframe Technical Exchange: October 3-5, 2023

Join us for the Mainframe Technical Exchanges virtually on October 3-5. Connect with Mainframe Experts who share the latest technical education and product demos and respond to your questions and feedback. These no-cost educational events are a great way to network with peers and experts from across the globe.
Bookmark this page for current registration and event information.

Custom Maintenance Acquisition Interface

Use the Create Service Order Online Interface to order, download, and receive maintenance packages for Broadcom mainframe product solutions. This option is more streamlined than using the SMP/E Internet Service Retrieval batch JCL option and requires no firewall changes.

Direct Product Acquisition into z/OSMF

A new GIMZIP download option is available to acquire a portable software instance from a secure Broadcom download server directly from z/OSMF. To determine if your product is GIMZIP enabled, see Mainframe Products using z/OSMF for Software Management.

Consolidated Product Lifecycle Overview

Use the new Broadcom Mainframe Product Lifecycle Page to determine the end-of-service (EOS) or end-of-life (EOL) support dates for Broadcom mainframe products and releases. This information is useful when planning installations and upgrades.

Chorus Software Manager (CSM) End of Life

As part of our ongoing commitment to customer success and to help our customer base achieve their strategic business initiatives, we are investing our resources in z/OSMF for software management. We discontinued technical support for CSM effective **June 30, 2023**. For full details, see the End of Life Announcement. We recommend that you install z/OSMF to manage your software. For installation and usage details, see the IBM documentation and the Broadcom product installation best practices at techdocs.broadcom.com.

Release Notes

Includes new feature descriptions, product compatibility details, and third-party software agreements.

The release notes explain the key features and details for REXX Db2 eXtensions Release 20.1.

RDX extends the REXX language with a native interface to Db2.

New Features

The new features in this RDX release offer you increased flexibility and efficiency. We release features using PTFs for simple installation.

RDX Release 20.1 provides the following enhancements. To ensure availability of all features and fixes, ensure that your products are current on maintenance. We recommend that you use the SMP/E Internet Service Retrieval to have maintenance that is downloaded and received automatically on your system regularly.

Transition to Common Components and Services

Beginning with this release, RAI Server is no longer required or supported. Common Components and Services (CCS) replaces the functionality that was previously provided by RAI Server. For information about installing this product, see the Installing section. For information about installing CCS, see the CCS documentation.

Resolved Issues

Review the resolved issues to understand the key fixes in this release.

This release includes the following resolved issue that was resolved in release 20.0:

- Relocating REXX IRX Initialization Parameter Modules (LU04261)

Relocating REXX IRX Initialization Parameter Modules (LU04261)

PTF LU04261 resolves the communication issue between Smart/RESTART and Smart/RRSAF, which resulted in Smart/RESTART and Smart/RRSAF batch job failures. Communication issues occur when batch jobs do not specify a STEPLIB or JOBLIB DD statement and the CRAIDPT library was added to a linklist or Link Pack Area (LPA).

By default, the IRXPARMS, IRXTSPRM, and IRXISPRM REXX initialization-parameter load modules are installed in the RAI product common CRAILOAD load library. The operating system then loads them during TSO region and REXX environment startup.

Applying this PTF moves the IRXPARMS, IRXTSPRM, and IRXISPRM REXX initialization parameter load modules to the CRAILRAI library.

This PTF is required to circumvent S66D abends in Smart/RESTART jobs executing in a IKJEFTxx TSO batch environment. However, even after applying the PTF, when RLX users run REXX executables that invoke any RLX service and only the CRAILOAD load library, and optionally Db2 DSNLOAD and DSNEXIT, are allocated in the STEPLIB, the RLX REXX executable fails with RC=-3 and error messages are displayed. For example:

```
* REXX error occurred at line 26
* EXEC IN ERROR......... RLXVSQL
* TYPE OF EXEC.......... COMMAND
* USER TOKEN............ ?
* ERROR CONDITION....... FAILURE
* REXX ERRROR........... -3
```

```
* ERROR MESSAGE......... NO ERROR TEXT AVAILABLE FOR RC = -3
* ERROR DESCRIPTION..... RLX DECLARE QUERY_REULT REXXSTEM FOR SELECT DISTINCT NAME, COLNO, COLTYPE, LENGTH
 ORDER BY COLNO
* ERROR IN SOURCE LINE#. 26
* Error source line:
* > "RLX DECLARE QUERY_RESULT REXXSTEM FOR"
```

RC= -3 indicates that a valid REXX language environment does not exist for executing the indicated REXX exec.

To circumvent this error, use one of the following methods:

- Edit the user REXX exec and add a CALL SDKINIT at the top and CALL SDKTERM at the end of the exec.
- Add the CRAILRAI library before the CRAILOAD library in the STEPLIB concatenation.

  NOTE
  The CRAILRAI and CRAILOAD libraries must be APF Authorized.

Product Names and Abbreviations

This documentation references the following products and abbreviations:

- REXX Db2 eXtensions (RDX)
- Common Components and Services (CCS)
- RAI
- RAI Server
- REXX Language Xtensions (RLX)
- TASKLIB+

Introduction and RDX Concepts

This documentation is a reference for REXX Db2 eXtensions (RDX). All references to SQL in this documentation imply the SQL that is associated with Db2 for z/OS. Similarly, all objects that are described are Db2 for z/OS objects. The syntax and semantics of most SQL statements are essentially the same as documented in the IBM Db2 publication *SQL Reference* for Versions 9, 10, and 11 of Db2, all of which are fully supported by RDX. Whenever required, the differences between RDX and DSNREXX (the REXX interface to Db2 provided by IBM) are discussed. Otherwise, readers can safely replace every reference to DSNREXX in the IBM publications with a substitute reference to RDX.

RDX fully supports all SQL and SQL/XML statements that are implemented in Db2 versions whose IBM support is current.

RDX is a program product that extends the REXX language with a native interface to Db2. RDX can be invoked in two modes: as a REXX command or through a REXX function call. RDX establishes a connection to Db2 using either RRSAF or CAF (Call Attach Facility). Both are standard Db2 attachment mechanisms. After connection to Db2 is successfully established, REXX programs can issue SQL statements and IFI calls. RDX supports the syntax of both dynamic and static SQL statements, with full support for embedded host variables. RDX seamlessly binds SQL host variables with REXX variables and provides the data conversion that is required by SQL statement semantics. A REXX program need not be pre-compiled and can be executed immediately after it is created. RDX sets the REXX variables SQLCODE, RC, and REASON code to communicate the results of SQL statement execution. Diagnostic messages can be displayed to help the developer better understand the cause of a problem.

Audience

This documentation is intended for application programmers, system administrators, and database administrators who want to develop applications quickly to access Db2 for z/OS. RDX uses the speed, ease, and expressive power of REXX by letting you embed SQL and SQL/XML statements natively within REXX programs. Such applications ordinarily take much longer to develop with compiled or assembled languages such as COBOL, PL/I, C, or Assembler.

Installing RDX

Installation information is relevant only if you are installing RDX as an independent product. Since RDX is now distributed with RLX, RDX is installed after the RLX installation is completed. You might only want to review RDX system defaults.

This installing information documents the RDX installation and describes the product libraries. The batch job streams that are described and illustrated must be modified in keeping with your installation conventions regarding dataset names, device types, and so on. Installation-dependent JCL parameters appear in this documentation as values within question marks **?likethis?** to make them easier to recognize. Modify these parameters before the job streams are submitted. For example, the **?raihlq?** variable designates the High-Level Qualifier that you assign to the RDX datasets when copying the distribution libraries to your system, for example, **RDX.V2R1M0**.

> **NOTE**
> See the $RDXREAD member of the restored CRAISAMP library for the values to be substituted for the ? question-mark-variables?. Some symbolic variables must be replaced with literal values exactly as defined in the $RDXREAD member while other symbolic variables must be replaced with site-specific values that you designate.

This list shows the RDX libraries that are installed during the SMP/E process:

- **CRAIDBRM**: Db2 database request modules
- **CRAIEXEC**: REXX Execs
- **CRAIJCL**: Release notes, installation JCL, and JCL to build and run sample applications
- **CRAILRAI**: The extended base load library, which also includes IRX REXX initialization modules
- **CRAILOAD**: The base load library, which also includes RAI Server load modules
- **CRAIMAC**: Macro library
- **CRAIMSG**: ISPF message library
- **CRAIPNL**: ISPF panel library
- **CRAIPROC**: JCL procedures
- **CRAISAMP**: Sample data for IVPs, assembler, C/C++, PL/1 program sources, and DBD gen for a sample IMS database
- **CRAISKL**: ISPF file tailoring skeleton library
- **CRAISQL**: Db2 DDL files
- **CRAIOPTN**: Various product customization and runtime options and parameters
- **CRAIHELP**: ISPF help library
- **CRAIPARM**: RAI Server parameter file

Pre-Installation Planning

Before you begin the installation process, identify the following information:

- The name of the Db2 plan to be used by RDX. The default plan name is of the form RDXPLAN. Every version of RDX uses several package collections named RDXvrmll -- where v, r, and m are the RDX Version, Release, and Modification level, respectively, and ll is the package isolation level: either CS, RR, RS, or UR.
- The name of the Db2 data base (and Table Spaces) in which the RDX demo tables are to be created.

In addition, the installer must have authority to BIND the Db2 application plan to be used by RDX.

Installation Summary

This article describes the RDX installation process.

Use standard SMP/E installation and maintenance best practices to install and maintain mainframe z/OS products. The installation process includes all the tasks that are typically performed by a systems programmer to acquire the products and make them ready for use in a production environment.

1. If Common Components and Services (CCS) is not already available, install CCS. For more information, see the CCS documentation.
2. Install the RAI products pax file. For the installation and configuration information, see the Smart/RESTART, RLX, and TASKLIB+ Installation documentation. After you complete the installation, return to this topic to complete the RDX configuration.
3. BIND the RDX plan in each Db2 subsystem in which RDX is to be installed.
4. Verify the RDX installation.
5. Deploy RDX for general usage.
6. Explore RDX functions by executing the sample REXX programs that are described in RDX Sample Programs.

BIND the RDX Plan

This article describes how to bind the RDX packages and the application plan.

To BIND the RDX packages and the application plan, use the JCL in the RDXJBIND member of the CRAIJCL library.

Follow the instructions that are embedded in the JCL of the **RDXJBIND** job to change the **?substitutable_variables?** that are used in the JCL with site-dependent values. Submit the BIND job and expect a jobstep completion code of 0000.

Several DBRM modules are found in the CRAIDBRM library, one for each supported Db2 version, named **RDXSQLn**, where **n** is Db2 version. Each DBRM contains a full set of SQL statements that are supported by the given Db2 version, and each DBRM is produced by the Db2 pre-compiler that is supplied with the respective Db2 version. The DBRM RDXSQL8 was prepared on Db2 Version 8 subsystem running in full-function mode.

The default name of the RDX plan is **RDXPLAN**. When you are binding RDX in a Db2 system where a prior version of RDX is already bound, you must add the RDX packages from previous RDX installations to the PKLIST of the RDXPLAN to allow new and previous RDX versions to be simultaneously operational in the Db2 system.

To allow RDX to be executed by the general user community, issue the following GRANT:

```
GRANT EXECUTE ON PLAN RDXPLAN TO PUBLIC;
```

Verify RDX Installation

Verify the RDX installation after the BIND of the RDX application plan runs successfully.

1. Modify member RDXSMP01 of the CRAIEXEC library to specify the name of the Db2 subsystem in which you installed RDX. Specify ssid by replacing these two lines:

   ```
   ADDRESS ISPEXEC 'VGET (rlxdsn)'
   ssid              = rlxdsn    /* => specify Db2 subsystem id */
   ```

   with this line:

   ```
   ssid = 'DSN'  /* => specify Db2 subsystem id */
   ```

   where ‘DSN’ is the name of your Db2 subsystem
2. In your TSO/ISPF session, exit to the TSO READY prompt and execute the following command. Replace **?raihlq?** with the High Level Qualifiers that are associated with the RDX target libraries.

   ```
   TSOLIB ACT DA('?raihlq?.CRAILRAI, ?raihlq?.CRAILOAD   ['?db2dsnload?'])
   ```

This TSOLIB command makes the CRAILOAD library and all its load modules accessible to your TSO/ISPF session through the standard MVS search order. Optionally, if the target DB2 subsystem DSNLOAD library was previously not added to the system LINKLIST, it should be included in the TSOLIB command.

3. At the TSO/E READY prompt, execute the following TSO command. Replace **?raihlq?** with the High Level Qualifiers that are associated with the RDX target libraries.

   ```
   ALTLIB ACT APPL(EXEC) DATASET('?raihlq?.CRAIEXEC')
   ```

   The ALTLIB command adds the CRAIEXEC library to the SYSEXEC concatenation so the RDX sample REXX programs (named RDXSMPxx) can be executed.
4. Invoke RDX by typing the following command at the TSO READY prompt:

   ```
   RDXSMP01
   ```

   This command executes the RDXSMP01 REXX program and produces output similar to the following example:

   ```
   =============== SAMPLE PROGRAM RDXSMP01  - DB2 SUBSYSTEM DSN  ===============

   >CNTL SYSTEM DSN         - SQLCODE = 0 RC = 0 REASON = 00000000
   --- CURRENT SQLID=USERID - Initially
   >SET CURRENT SQLID        - SQLCODE = 0 RC = 0 REASON = 00000000
   --- CURRENT SQLID=PAYROLL  - After SET PAYROLL in a Host Variable
   >SET CURRENT SQLID        - SQLCODE = 0 RC = 0 REASON = 00000000
   --- CURRENT SQLID=USERID - After SET USER
   >SET CURRENT SQLID        - SQLCODE = 0 RC = 0 REASON = 00000000
   --- CURRENT SQLID=PAYROLL  - After SET PAYROLL as a literal
   >SET CURRENT SQLID        - SQLCODE = 0 RC = 0 REASON = 00000000
   --- CURRENT SQLID=USERID     - After SET SESSION_USER
   >TERM                     - SQLCODE = 0 RC = 0 REASON = 00000000  - SQLCODE = 0 RC = 0 REASON = 00000000
   READY
   ```

If the execution of RDXSMP01 was successful, then RDX is installed in a given Db2 subsystem and it is ready to be deployed. You can use this procedure each time you install a new RDX version and want to validate its installation.

> **NOTE**
> The validation procedure uses the default RDX plan name, RDXPLAN. If you changed the default RDX plan name (for example, to MYPLAN), you must modify the CRAIEXEC program as follows:
>
> ***From:***
>
> ```
> 000023 defaults = 'SYSTEM('ssid')'
> ```
>
> ***To:***
>
> ```
> 000023 defaults = 'SYSTEM('ssid') PLAN(MYPLAN)'
> ```

Deploy RDX for General Usage

This article explains how to deploy RDX for general use.

1. Verify RDX Installation described one method of including the CRAILOAD library in the program search order for a TSO/ISPF session. Alternatively, enable the system to find the load modules residing in the CRAILOAD library by some other method. Some viable alternatives include the following items:
   – Include the CRAILRAI and CRAILOAD libraries in the MVS LINKLIST using the PROGxx member of PARMLIB member. Specify control statements as shown in the following example:

     ```
     LNKLST ADD NAME(LNKLST00) DSN(?hlq?.CRAILRAI) VOLUME(?volser?)
     LNKLST ADD NAME(LNKLST00) DSN(?hlq?.CRAILOAD) VOLUME(?volser?)
     ```

     Issue an MVS SETPROG LINKLIST command to dynamically add the CRAIRAI and CRAILOAD libraries to LINKLIST (without the need to IPL the system).

To enable full RDX functionality, execute the following z/OS command:

```
SETPROG LPA,ADD,MODNAME=(IRXPARMS,IRXISPRM,IRXTSPRM)
  DSNAME=hlq.CRAILRAI
```

To reverse the effect of this command, execute this command:

```
SETPROG LPA,DEL,MODNAME=(IRXPARMS,IRXISPRM,IRXTSPRM)
```

Doing so enables access to RDX Host Command and RDX REXX functions globally.

- Place the CRAILRAI and CRAILOAD datas sets in the STEPLIB concatenation of the TSO/ISPF LOGON procedure. In this case, the CRAILOAD data set must be APF authorized. Doing so is a TSO requirement.
- If RDX is to be used only within ISPF, place the CRAILRAI and CRAILOAD libraries in the ISPLLIB concatenation. In this case, CRAILRAI and CRAILOAD do not need to be APF authorized.

2. Use the CRAIEXEC sample programs as patterns for your own REXX routines that use the RDX services. RAI suggests you copy the RDX sample routines into your own library that is allocated to the SYSPROC or SYSEXEC file. Then alter the sample REXX programs as required.

RDX sample programs may contain user-defined REXX functions in addition to SQL and SQL/XML statements. These user-defined REXX functions require special enablement. These functions are documented in the Software Developer Workbench (SDW) product. To execute sample RDX programs that employ these functions, use the SDWB program as a front end, which creates the necessary environment for these functions. For example, from TSO READY, enter the following command:

```
SDWB RDXSMP02
```

The sample program RDXSMP02 calls a function that is named REXXVARS. This function is not found if the SDWB program is not used as a front end.

RDX System Defaults

This article explains RDX system defaults.

After you establish a pattern of RDX usage, you might want to change some of the RDX global defaults that are defined in the RDX$TSD CSECT. The following sample shows the RDX system defaults:

```
_____________________________________________________________________

RDX$TSD  CSECT
RDX$TSD  RMODE ANY
RDXVRM   DC    CL4'130A'     RDX Version Release Modification Level
RDXSSNM  DC    CL4'XXXX'     Subsystem name
RDXPLAN  DC    CL8'RDXPLAN ' Plan name
RDXMAS#  DC    AL4(102400)   maximum size of STOR block
RDXVPREF DC    CL4'RDXV'     Variable prefix (non-table columns)
RDXSTART DC    F'1'          First row to fetch in a Result Set (REXXSTEM)
RDXLIMIT DC    F'999999'     Max number of rows to fetch        (REXXSTEM)
RDXERROR DC    C'R'          CONTROL ERRORS: R=RETURN, C=CANCEL
RDXMSUPP DC    C'N'          message suppression lvl: T|V|D|N
RDXMDEST DC    C'T'          message destination: F|T|W|N
RDXLOCT  DC    C'B'          Locator type: B|C|D|R - Used by HOLD/FREE
RDXXMLDT DC    H'457'        default XML as LONG VARCHAR CHAR with NULL
RDXDFDT  DC    H'480'        default substitute for DECFLOAT = FLOAT
RDXXMLLN DC    F'32760'      default maximum XML datatype length
RDXACEM  DC    C'Y'          Auto Create Error Msgs for SQLCODE<>0
RDXVER   DC    C' '          Db2 version
RDXRLX   DC    C'Y'          RLX compatibility mode
RLXDEBUG DC    B'00000000'   Debug RDX invocation session
```

```
RDXMAXL# DC    A(1024*1024)  max LOB size - truncate if it is greater
RDXMAXL# DC    A(1024*256)   max LOB size - truncate if greater
RDXAXP   DC    C'Y'          ATPROMPT(Y|N)
RDXITL   DC    C'D'          ISPF table load control of ISPFTABEL serv.
*                            X=eXtend - do not create a new ISPF table
*                                       and use TBADD to add new rows
*                            D=Delete - delete ISPF table then load a
*                                       new result set
RDXURC   DC    C'#'          Underscore Replacement Character/ISPFTABL
RDXEPAN  DC    C'N'          CNTL EPANEL NONDISPL
RDXCMP   DC    C'Y'          Compatibility with MAX/REXX
*
         DC    XL3'000000'   * vacant *
         END   RDX$TSD
```

You can change this CSECT using the IMASPZAP program, under the supervision of Broadcom Support. The RDX$TSD CSECT is included in load module RDX$TSD.

Changing RDX System Defaults – Job RDXJTSD

You can use job *hlq*.CRAIJCL(RDXJTSD) to change the RDX system defaults.

The following example shows job RDXJTSD that changes RDX system defaultsC

```
. . .
//jobname JOB
//*   RDXVRM(211A)       - RDX VRM
//*   SYSTEM(DSN)       - Db2 subsystem name
//*   PLAN(rdxplan)      - Db2 plan name
//*   MAXSTOR(102400)    - Maximum size of STOR block
//*   VARPREF(RDXV)      - Variables prefix for non-table columns
//*   START(1)           - First row to fetch in a result set
//*   LIMIT(999999)      - Maximum number of rows to fetch
//*   ERRORS(R|C)        - CONTROL ERRORS RETURN | CANCEL
//*   MSGSUPP(N|V|T|D)   - Message suppression level
//*   MSGDEST(N|F|T|V|R) - Message destination
//*   LOCTYPE(B|C|D|R)   - Locator type used HOLD/FREE
//*   XMLDTYPE(457)      - Default XML datatype - LONG VARCHAR NULL
//*   DECFLDT(480)       - Default for DECFLOAT = FLOAT
//*   XMLDTLEN(32760)    - Default maximum XML datatype length
//*   AUTOERRM(A|Y|N)    - Auto create error message
//*   DEBUG(N)           - Debug mode
//*   MAXLOBSZ(1024*256) - Maximum LOB size - truncate if greater
//*   ATPROMPT(N|Y)      - ATPROMPT(Y|N) attention exit prompt
//*   ISPFTBLC(D|X)      - ISPF table load control: Delete | Extend
//*   URC(#)             - ISPFTABLE Underscore Replacement Char
//*   EPANEL(N|Y)        - CONTROL ERRORS DISPLAY | NONDISPL
//*   COMPAT(N|Y)        - MAX/REXX compatibility
//*
//DEFAULT EXEC PGM=IRXJCL,
// PARM='RDXJTSD PLAN(RLX916CS) COMPAT(Y)'
//STEPLIB  DD  DSN=&RAIHLQ..CRAI.LOAD,DISP=SHR
//SYSLIB   DD  DSN=&RAIHLQ..CRAI.LOAD,DISP=SHR
```

```
//SYSEXEC  DD  DSN=&RAIHLQ..CRAI.EXEC,DISP=SHR
//SYSIN    DD  DSN=&&SPZAPIN,DISP=(,PASS,DELETE),UNIT=SYSDA,
//             SPACE=(TRK,(1,1)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000)
//SYSPRINT DD  SYSOUT=*
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  DUMMY
```

Valid parameters that are used in the job RDXJTSD are documented in the JCL. This example demonstrates changing the Db2 application plan to RLX916CS and setting MAX/REXX compatibility mode to ‘Y’.

Run-Time Control Service – CNTL

The CNTL command is the RDX control service that is used to configure RDX session parameters and to request RDX diagnostic services. The syntax of the CNTL command is:

```
"CNTL cntl_parameters"
```

You can specify cntl_parameters in any order and in any combination. CNTL parameters may be specified individually, as in 'CNTL SYSTEM Db2T'. Alternatively, several parameters can be specified together in the *same* CNTL command, as in 'CNTL SYSTEM Db2T PLAN RDXPLAN'.

COMPAT Y | N

The COMPAT command (stands for compatibility) sets the field RDXCMP in the RDX$TSD table of parameter defaults. This parameter is used to indicate to RDX to perform processing compatible with similar ISV products, such as MAX/REXX. For example, the parameter triggers the auto-release of allocated cursor-related statements, thus removing the need to issue the explicit "RELEASE Cn" commands. Exercise caution when using this command to avoid unintended consequences.

DEBUG {SQL|APPLICATION|ISPFTABLE|MEMORY|ALL | NO}

Activate the RDX debugging module to produce various levels of problem diagnostic output. Typically, you use this command when requested by Broadcom Support. You can specify the following options, one at a time, in order of increasing detail of output:

- SQL – Display all commands (including SQL statements) passed to RDX
- APPLICATION – Display certain application events to be used in problem diagnosis
- ISPFTABLE – Display all ISPFTABLE service-related activities
- MEMORY – Display memory management activities
- ALL – Display all diagnostic information except memory activities which must be started by an individual DEBUG command.
- NO – Disable of active debugging options

Example: Activate all available debugging traces

```
"CNTL DEBUG ALL DEBUG MEMORY"
```

DSNTIAR { CALL | AUTO | NO | YES }

The IBM-supplied DSNTIAR program is used to translate information in the SQLCA into a meaningful Db2 message. The following values may be specified for the DSNTIAR parameter of the RDX CNTL service:

- YES - directs RDX to always invoke DSNTIAR and to externalize SQLERR.i stem variables, regardless of SQLCODE. This option is a system default.
- AUTO - directs RDX to call DSNTIAR after every SQL statement that returns non-zero SQLCODE. This option provides the most efficient RDX operation. If you must externalize the contents of SQLCA, for example, after a

successful UPDATE SQL statement to find out the number of rows that are affected by the UPDATE statement that is placed in SQLERRD.3 variable, use the “CNTL DSNTIAR CALL” statement in these exceptional cases.

- CALL - directs RDX to call the DSNTIAR routine immediately with SQLCA that is associated with the latest SQL call. RDX then creates a stemmed array that is named SQLERR.i and other SQLCA variables, such as SQLERRD.j stem, which your REXX routine can access. This information can also be obtained through a GET DIAGNOSTIC statement.
- NO - directs RDX never to call DSNTIAR, regardless of the SQLCODE that is received after an SQL statement. Only the SQLCODE variable is externalized. To populate SQLERR.i stem variables with the latest SQLCA data, execute “CNTL DSNTIAR CALL” statement.

EPANEL DISPLAY | NONDISPL

The EPANEL (Error Panel) command directs RDX to DISPLAY or NONDISPL (not display) an error panel after an SQL statement has resulted in a non-zero SQLCODE. The DISPLAY function may be useful when you want to see the error completion of SQL statements without coding REXX statements to examine SQLCODE and SQLERR.i stem variables. The default is NONDISPL.

ERRORS { RETURN | CANCEL }

The CNTL ERRORS parameter directs RDX to respond to errors that produce negative SQL codes as follows:

- RETURN - after a negative SQLCODE, return control to the user’s REXX program.
- CANCEL - after a negative SQLCODE, terminate the user’s REXX exec through a TSO/REXX Halt Interpretation command.

ISPFTABLE DELETE | EXTEND

The ISPFTABLE command defines an action to be performed when DECLARE ISPFTABLE services are executed and the ISPF table to be created already exists:

- DELETE - replace the existing ISPF table with a new structure and result set.
- EXTEND - add rows of a new result set to the existing ISPF table.

LIMIT nnnnnn

The LIMIT parameter pertains to the operation of the DECLARE REXXSTEM and DECLARE ISPFTABLE statements. The parameter sets a limit on the number of result set rows that RDX returns to the calling EXEC. See the related START parameter for a further discussion. The default value is 999999.

> **NOTE**
> Instead of using the LIMIT command, you can use the Db2 “FETCH FIRST nnnnnn ROWS ONLY” clause on the SELECT statement.

PLAN plan_name

The PLAN parameter sets the name of the Db2 application plan to be used by RDX. This option is only valid *before* the RDX application thread to Db2 is established. The default plan name is RDXPLAN.

PREFIX pppp

The PREFIX parameter specifies up to four characters to be used as a prefix in forming a variable name to be associated with a result set column that has no implied column name. The default prefix is ‘RDXV’, which generates variable names such as RDXV001, RDXV002, RDXV003.

Example:

```
"EXECSQL DECLARE C1 REXXSTEM FOR",
 "SELECT 'Hello World!' INTO SQLNAME",  "FROM SYSIBM.SYSDUMMY1"
```

This statement returns two variables:

```
C1.ROWS = 1
C1.RDXV001.1 = 'Hello World!'
```

RELEASE Sn | ALL

The CNTL RELEASE parameter directs RDX to release all control blocks that are associated with an individual statement Sn (n = 1 … 200) or ALL statements. Doing so also frees all the associated cursors Cn. Use the CNTL RELEASE command to free prepared statements if you must re-execute them and you have no need for the previously prepared SQL statement. For this command to be successful, the associated cursor Cn (if it was defined) must be closed. The ALL keyword directs RDX to free *all* prepared statements.

SNAP

The SNAP parameter directs RDX to print its internal control blocks for problem diagnosis. Execute this command only when requested by Broadcom Support.

START nnnnnn

The START parameter pertains to the operation of the DECLARE REXXSTEM and ISPFTABLE statements. START directs RDX to skip first *nnnnnn* rows in the result set. Thus, row number nnnnnn+1 becomes the first row of the result set that RDX returns in the REXX stem. See the related LIMIT parameter for further discussion. The default value for the START parameter is 1.

SYSTEM ssid

The SYSTEM parameter specifies the name of the Db2 subsystem to which RDX should connect. The SYSTEM parameter is only valid before a RDX application thread to Db2 is established (explicitly or implicitly) because it defines the Db2 subsystem with which RDX establishes communication.

VERSION version

The CNTL VERSION parameter specifies a Db2 version that is less than or equal to the version of the Db2 subsystem to which RDX is to connect. The VERSION parameter directs RDX to load the module named RDXSQLx, where x is equal to the version value that is specified by the VERSION parameter.

Use the VERSION parameter when RDX application processing may involve distributed activity on remote subsystems running different versions of Db2. For example, a REXX program may connect to a local subsystem that is running Db2 Version 9 and may issue a distributed request that executes on a remote subsystem running Db2 Version 8.

Select a value for VERSION that is the *least common denominator* among all versions of Db2 subsystems that are involved in the communication. The allowed values for VERSION are 8 and 9. Specify a VERSION value of 9 for Db2 V10 since there is no new static SQL introduced in Db2 V10. Note that the VERSION option is only valid *before* the RDX application thread to Db2 is established. The default value is the version of the Db2 subsystem to which RDX connects initially.

Example:

The following CNTL command must be issued *before* any other RDX statements are executed. Doing so sets the following default parameters:

- Assigns DSN as the Db2 subsystem name for an implicit Db2 connection
- Directs RDX to halt the execution of the REXX program when a negative SQLCODE is received
- Directs RDX to use the SQL statements embedded within the RDXSQL8 load module. Doing so enables SQL statements to execute on a remote subsystem running Version 8 of Db2.

```
ADDRESS RDX "CNTL SYSTEM DSN ERRORS CANCEL VERSION 8"
```

NOTE
Starting with Db2 V10, it is no longer necessary to maintain different versions of the RDXSQLx module since all static SQL statements supported by RDX have been synchronized and placed in the RDXSQL module. Usage of the VERSION command is now unnecessary. The VERSION command is retained for compatibility.

XMLDT nnn

The XMLDT parameter specifies the default data type to be used for XML columns. In Db2 9 and later, columns that are defined with the native XML data type cannot be stored or retrieved directly. These columns must instead be *cast* to one of the following data types (and SQLTYPEs): VARCHAR (456), BLOB (404), BLOB (408), or DBLOB (412). Specify nnn as one of the SQLTYPE values listed, or add 1 if a result column may assume a value of NULL. The default is 457 (VARCHAR with NULLs).

XMLLEN nnn

The XMLLEN parameter specifies the default length to be used for XML columns (as described in the XMLDT parameter). The default length is 32760.

Establishing and Terminating the Connection to Db2

Before you can start issuing SQL and/or SQL/XML statements from a REXX exec using RDX commands, you must establish a REXX-to-Db2 connection, or, in Db2 parlance, create an *application thread*. At the completion of processing, you must disconnect from Db2. To avoid connectivity problems, complete the following checklist:

- Make sure that RDX is installed and the RDX application plan was bound. The default plan name is RDXPLAN.
- Arrange for the CRAILRAI and CRAILOAD libraries to be available to your TSO/ISPF session. To do so, place the CRAILRAI and CRAILOAD libraries in the concatenations for LINKLIST, STEPLIB, or ISPLLIB.

Connecting to the Db2 Subsystem

RDX can use two methods to establish a connection to Db2: *implicit* or *explicit* connection methods. RDX can either CONTINUE or CANCEL REXX program execution should RDX fail to connect to a Db2 subsystem. The default setting is CNTL ERRORS CONTINUE, so your REXX routine continues to execute. This behavior enables the REXX routine to display the SQLCODE, RC, and REASON variables that are associated with the Db2 connection failure as well as the SQLERR.i stem, which RDX populates with error messages from Db2.

Implicit Connection to Db2

RDX establishes an implicit connection to Db2 when an EXECSQL command is issued and no Db2 application thread is presently active. Before establishing an implicit connection, RDX checks if any of the following REXX variables are defined in your REXX exec:

REXX Variable	Meaning
_Db2	Name of the target Db2 subsystem (the default is DSN).
_PLAN	Name of the RDX plan to use to establish the thread (the default is RDXPLAN).
_VERSION	Value of the Db2 version (7, 8, or 9) to be used for backward compatibility when processing for an application thread is distributed across multiple Db2 subsystems running different versions of Db2 code. The use of this parameter is discussed in the documentation for the CNTL VERSION.

You can typically accept the default plan name of RDXPLAN, so the _PLAN variable can generally be omitted. However, the name of the Db2 subsystem to which RDX should connect is *not* usually the default name DSN. As such, the _Db2 variable should *always* be specified explicitly. RDX establishes a connection with the Db2 subsystem that is specified by the value of the _Db2 variable. Alternatively, you can use the CNTL SYSTEM ssid command to set the default Db2 subsystem name.

If the implicit Db2 connection fails and the REXX program continues execution (for example, CNTL ERROR CONTINUE is in effect), all subsequent SQL statements fail with SQLCODE -924. If you then wish to connect *explicitly* to a given Db2 subsystem, you first must issue a RDX TERM command to terminate the failed implicit connection. Only then can you issue another INIT call to explicitly connect to a named Db2 subsystem.

Example:

```
/* REXX */
ADDRESS RDX
_Db2 = 'DSN'
"EXECSQL SET CURRENT SQLID = 'PAYROLL'"
SELECT
```

```
  WHEN SQLCODE = -924 THEN SAY 'Failed to connect to Db2' _Db2
  WHEN SQLCODE <> 0   THEN SAY 'Bad SQLCODE =' sqlcode
  OTHERWISE; NOP
END
EXIT
```

Explicit Connection to Db2

Establish an explicit connection to Db2 by issuing a RDX INIT command.

To connect to Db2 explicitly, issue a RDX INIT command like the following example:

```
ADDRESS RDX
"INIT parameter_list"
```

The parameter_list is a list of parameters, each of which has the general format: keyword(value). Parameters that allow only YES or NO values can be specified without a value. The absence of a value denotes a YES.

Keyword	Values	Default	Attachment
SYSTEM	Name of the Db2 subsystem.	DSN	CAF RRSAF
PLAN	Name of the Db2 application plan.	RDXPLAN	CAF RRSAF
COLLID	Collection ID to be initially associated with the RDX application thread.	Assigned by Db2	RRSAF
PAUTHID	Primary authorization ID to be assigned to the RDX application thread.	Assigned by Db2	RRSAF
SAUTHID	Secondary authorization ID to be assigned to the RDX application thread.	Assigned by Db2	RRSAF
CORRID	Correlation ID assigned to the application thread.	Assigned by Db2	RRSAF
ACCT	Accounting token. Use this token to identify the thread in an accounting IFCID.	Assigned by Db2	RRSAF
APPL	Application or transaction name.	Not assigned by Db2 for z/OS	RRSAF
USER	User description that is associated with the RDX application thread.	Not assigned by Db2 for z/OS	RRSAF
WSN	Workstation name.	Not assigned by Db2 for z/OS	RRSAF
CONTEXT	Used by RRSAF internal logic.	Presently ignored	RRSAF
PGMID	Program identification.	Not assigned by Db2 for z/OS	RRSAF

Keyword	Values	Default	Attachment
SQLMON	Controls invocation of the SQL Monitor with one of the following options: FULL – In addition to DETAIL reports, also formats SQLDA for every SQL call. DETAIL – In addition to SUMM reports, also displays the RDI information for every executed SQL statement. SUMM - Prints only summary of SQL statements execution that shows the total number of SQL statements by type (for example, OPEN, FETCH, and so on). OFF - Disables SQL Monitor. To format DSNHDECP Db2 defaults, use the MSGSUPP(DEBUG) parameter.	OFF	N/A
SQLERROR	Directs RDX to call (YES or NO) the IBM supplied DSNTIAR routine to translate the SQLCA and print error messages when the SQLCODE is not zero: This parameter applies only to SQLMON reports.	NO	N/A
ATTACH	Db2 attachment to be used: CAF or RRSAF.	CAF	CAF RRSAF
TERMACT	Thread termination action: COMMIT, ROLLBACK, AUTO, or NONE COMMIT directs RDX to always issue a COMMIT request at the thread termination. ROLLBACK directs RDX to issue a ROLLBACK at the thread termination. If the ROLLBACK default is accepted, then explicit SQL COMMIT statements must be issued to harden any changes. AUTO causes a COMMIT to be issued when the last SQLCODE is zero. Otherwise issue a ROLLBACK. When NONE is specified, neither a COMMIT nor a ROLLBACK is issued at the thread termination.	ROLLBACK	N/A
ERRORS	Governs the response when RDX detects a negative SQLCODE. RDX should either RETURN control to the active application thread or ABEND.	RETURN	N/A
MSGDD	Specifies the DDNAME of the message file.	USQPRINT	N/A
MSGDEST	Specify an output destination for messages: FILE – write to the file allocated to the MSGDD(ddname) TSO – output messages through TPUT to the TSO console WTO – write messages to the operator through WTO NONE – no messages is issued	NONE	N/A

Keyword	Values	Default	Attachment
MSGSUPP	Message suppression level: TERSE – issue only E-level messages VERBOSE – issue E, W, and I-level messages DEBUG – issue all messages NONE – suppress all messages	NONE	N/A

Example 1: Connect to the Db2 subsystem named DSN using the RRSAF attachment:

```
"INIT SYSTEM(DSN) ATTACH(RRSAF)"
```

Example 2: Connect to the Db2 subsystem named DSN using the default CAF attachment and print the SQL Monitor report in FULL mode:

```
"INIT SYSTEM(DSN) SQLMON(FULL)"
```

> **WARNING**
> If the INIT command fails, first issue a RDX TERM command to terminate the failed environment, *before* you issue a new INIT command.

Example 3: Attempt to connect to the Db2 subsystem named DSN. If the connection attempt fails, issue a TERM command to terminate the failed connection. Issue another INIT call to connect to the Db2 subsystem named DSN.

```
"INIT SYSTEM(DSN) SQLMON(FULL)"
IF SQLCODE \= 0 THEN DO
"TERM"        /* terminate failed Db2 connection */
"INIT SYSTEM(DSN)"
END
```

Disconnecting from Db2 Subsystem

You can use two methods to terminate a connection to a Db2 subsystem: *implicit* and *explicit* termination. To explicitly terminate a connection to a Db2, issue a RDX TERM command like the following example:

```
ADDRESS RDX "TERM [SYNC|ABRT|NONE])"
```

The thread termination processing implies two processing modes: COMMIT and ROLLBACK. When you specified a command:

- **"TERM SYNC"** - A COMMIT is issued by Db2 before the Db2 thread is disconnected.
- **"TERM ABRT"** - A ROLLBACK is issued by Db2 before the Db2 thread is disconnected.
- **"TERM NONE"** - Neither a COMMIT nor a ROLLBACK SQL statement is issued; effectively RDX does not explicitly disconnect from Db2. This command is used in REXX stored procedures.
- **"TERM"** - The thread termination mode is affected by TERMACT parameter.

When your REXX exec that issued the SQL statements terminates without issuing an explicit "TERM" command, the termination processing is performed by Db2 (not RDX). Db2 monitors active application threads and is being notified by z/OS when an application TCB (a Task Control Block), under which the Db2 connection was established, terminates. When this behavior happens, Db2 issues a ROLLBACK statement before the application thread is terminated.

> **NOTE**
> When the first REXX exec is invoked in the TSO/ISPF environment, the process is done with the TSO EXEC command. The latter always creates a TCB under which the REXX exec executes. If your REXX exec issues a REXX CALL statement, the target REXX exec is invoked under the caller's TCB. You should consider this behavior when omitting an explicit "TERM" command while using a hierarchy of REXX execs, one calling another.

The recommended practice is to issue in your REXX program an explicit COMMIT or ROLLBACK SQL statement to avoid ambiguity in the termination actions. The exception is usage of the “TERM NONE” command in REXX stored procedures, where no thread termination occurs when the REXX exec terminates.

SQL Statements

This section reviews all the Db2 for z/OS SQL statements that RDX can execute. RDX implements the syntax of Db2 SQL statements exactly as documented by the Db2 for z/OS publications. However, in some instances, specific parameters of specific SQL statements are *not* supported by RDX. These discrepancies are documented in the descriptions of SQL statements.

RDX also provides the DECLARE REXXSTEM statement as an extension to Db2 SQL. DECLARE REXXSTEM is implemented as a composite of the SQL DECLARE, OPEN, FETCH, and CLOSE statements. DECLARE REXXSTEM statements materialize SQL query results and map them into an array of REXX stemmed variables.

How SQL Statements Are Invoked

SQL statements are invoked from a REXX exec using a RDX command with the EXECSQL prefix.

Doing so makes the RDX SQL statements compatible with the DSNREXX syntax and also serves to distinguish them from RDX INIT and TERM commands and IFI requests. An Address RDX statement instructs REXX to direct host commands to the RDX Host Command Environment:

```
ADDRESS RDX
"EXECSQL sql_statement_text"
```

Alternatively, RDX can be invoked as a REXX function:

```
rc = RDX("EXECSQL sql_statement_text")
```

The *sql_statement_text* is any syntactically valid SQL statement that may be executed in the Db2 for z/OS environment. Except for quoted strings, SQL statements are case-insensitive, so you can make them expressive using upper and lower case characters. Character strings that are enclosed in single or double quotes are case sensitive and are not converted to uppercase.

The IBM publication *Db2 SQL Reference for IBM Supported Releases* is essential for the understanding and proper usage of SQL statements, and should always be referenced as a companion to this documentation.

All SQL statements that are supported by Db2 for z/OS are listed and discussed. To avoid repetition, only information that is unique to RDX is presented. The syntax and usage of SQL statements should be obtained from the SQL Reference for a given Db2 version.

ALLOCATE CURSOR

RDX support for the ALLOCATE CURSOR statement is limited to a predefined set of cursors -- namely C101, C102, … C200. For example:

```
ALLOCATE C101 CURSOR FOR RESULT SET :rs_locator
```

If you specify a cursor name other than the predefined range between C101 and C200, the command fails with a RDX error message.

Example:

```
procname          = 'RDX.RDBTSP1'
table             = 'RDX.DEMO'
rank              = 5
"EXECSQL CALL :procname (:table,:rank,:msg)"
"EXECSQL DESCRIBE PROCEDURE :procname INTO :out"
"EXECSQL ALLOCATE C105 CURSOR FOR RESULT SET :out.1.sqllocator"
```

In this example, a stored procedure whose name is stored in the variable procname is successfully executed (through a SQL CALL) and described into the SQLDA stored in the REXX stem out. The out.1.sqllocator variable contains the value of a result set locator that is allocated as a cursor that is named C105 (within the range of C101 to C200). Subsequently, the cursor C105 can be FETCH-ed and CLOSE-ed like a regular cursor. See the sample program in the RDXSPM05 member of the CRAIEXEC library for a working example.

ALTER

RDX executes any syntactically correct ALTER statement. You can specify an ALTER statement directly, like so:

```
"EXECSQL ALTER . . ."
```

RDX executes the ALTER statement by issuing an EXECUTE IMMEDIATE SQL statement internally. As an alternative, you can execute an ALTER statement using an EXECUTE IMMEDIATE statement like the following example:

```
statement = 'ALTER TABLE RDX.DEMO . . ."
"EXECSQL EXECUTE IMMEDIATE :statement
```

Consult the sample REXX program in the RDXSMP07 member of the CRAIEXEC library for a working example.

ASSOCIATE LOCATORS

RDX supports the ASSOCIATE LOCATORS statement, but with support for only one locator host variable within the LOCATOR list, as follows:

```
ASSOCIATE LOCATOR (:rs_locator) WITH PROCEDURE :procname
```

In cases where a stored procedure returns more than a single result set, use the DESCRIBE PROCEDURE statement *instead of* the ASSOCIATE LOCATOR statement. The DESCRIBE PROCEDURE statement returns all the locators that are associated with a stored procedure execution instance into the REXX stem that is specified by the DESCRIBE issuer as the SQLDA.

Example:

```
"EXECSQL CALL :procname (:parm1, :parm2)"
"EXECSQL DESCRIBE PROCEDURE :procname INTO :out"
DO i = 1 to out.SQLD
   SAY out.i.sqllocator
END
```

In this example, the DESCRIBE PROCEDURE statement fills the REXX stem named out with all the result set locators that are returned by the stored procedure.

Consult the sample REXX program in the RDXSMP05 member of the CRAIEXEC library for a working example.

BEGIN DECLARATION SECTION

RDX does not support this statement.

CALL

RDX fully supports the SQL CALL statement. You can prepare parameters for the CALL statement in the following ways:

- Initialize all fields of a REXX stem that represents the SQLDA to be used as the input to the CALL statement. For example:

  ```
  CALL :procname USING DESCRIPTOR :stem
  ```

This method may be convenient to use when you retrieve information about the stored procedure from the Db2 catalog. You must be careful to initialize all fields in the SQLDA. Consult the sample REXX program in the RDXSMP08 member of the CRAIEXEC library for a working example.

- Use DECLARE VARIABLE statements to define all the parameters of the stored procedure *before* calling it, as in:

  ```
  CALL :procname (:parm1,:parm2,…).
  ```

  Consult the sample REXX program in the RDXSMP05 member of the CRAIEXEC library for a working example.

CLOSE

The SQL CLOSE cursor statement is fully supported by RDX.

Example:

```
"EXECSQL CLOSE C5"
```

COMMENT

RDX fully supports the COMMENT statement, with support similar to the support described in the ALTER statement.

Example:

```
"EXECSQL COMMENT ON TABLE RDX.DEMO IS",
 "'RDX DEMONSTRATION TABLE'"
```

COMMIT

RDX supports the SQL COMMIT statement as well as the CPIC RRSCMIT function. The syntax of the RDX COMMIT statement is as follows:

```
COMMIT {WORK} | {CPIC}
```

When the RRSAF attachment is used, instead of CAF, a COMMIT can also be requested through the CPIC RRSCMIT function as follows:

```
COMMIT CPIC
```

Use the CPIC form of COMMIT if:

- Your application is using the RRSAF attachment – INIT ATTACH(RRSAF)
- Your application accesses other RRS-compliant resources such as WebSphere MQ in addition to Db2. In this case, use the CPIC form of COMMIT to avoid having Db2 abnormally terminate your application.

If the RRSCMIT call fails, you receive a RDX102 error message showing the return code value. For an explanation of the return code, see the IBM publication *z/OS MVS Programming: Callable Services for High-Level Languages*.

All considerations discussed here also apply to the ROLLBACK statement.

Consult the sample REXX program in the RDXSMP09 member of the CRAIEXEC library for a working example. RDXSMP09 illustrates various forms of the COMMIT statement.

CONNECT

The CONNECT statement is fully supported by RDX. For the statement syntax, see *Db2 SQL Reference* publications. RDX maps the CONNECT statement within your REXX exec to one of the following static SQL statements:

```
CONNECT TO :SERVER
CONNECT TO :SERVER USER :USER USING :PASSWORD
CONNECT RESET
```

```
CONNECT
CONNECT USER :USER USING :PASSWORD
```

Information about the server is placed in the REXX variable that is named SQLERRP. The information appears in the form *pppvvrrm*, where:

- *ppp:*
  - is ARI for Db2 Server for VSE and VM
  - is DSN for Db2 for z/OS
  - is QSQ for Db2 for i5/OS
  - is SQL for Db2 for LUW
- *vv* - is a two-digit version identifier such as '09'
- *rr* - is a two-digit release identifier such as '01'
- *m* - is a one-digit maintenance level. Values 0, 1, 2, 3, and 4 are reserved for maintenance levels in compatibility and enabling-new-function mode. Values 5, 6, 7, 8, and 9 are for maintenance levels in new-function mode.

For example, if the server is Version 9 of Db2 for z/OS in new-function mode with the first level of maintenance, the value of SQLERRP is 'DSN09015'.

Example:

```
"INIT SYSTEM(DSN8)"
location = 'RADSN7'
CONNECT TO :location
SAY 'Server information' SQLERRP
```

Consult the sample REXX program in the RDXSMP04 member of the CRAIEXEC library for a working example.

CREATE

All forms of the CREATE statement are supported. See the description of the ALTER statement for further discussion.

Example:

```
"EXECSQL",
"CREATE TABLESPACE RDXDEMO IN RDXTEST",
  "USING STOGROUP RDXTEST PRIQTY 25 SECQTY 500",
  "ERASE NO",
  "FREEPAGE 0 PCTFREE 5",
  "GBPCACHE CHANGED",
  "BUFFERPOOL   BP0",
  "LOCKSIZE     ANY",
  "LOCKMAX      0",
  "CLOSE        YES"
```

Consult the sample REXX program in the RDXSMP06 member of the CRAIEXEC library for a working example.

CREATE TYPE(array) (Db2 V11)

A new CREATE TYPE(array) statement that was introduced in Db2 V11 is fully supported by RDX. For example:

```
CREATE TYPE PHONENUMBERS AS DECIMAL(10,0) ARRAY[50]
```

You can replace the square brackets with the following format: '??('='[', '??)'='']'

So that the preceding example can be rewritten as follows:

```
CREATE TYPE PHONENUMBERS AS DECIMAL(10,0) ARRAY??(50??)
```

Doing so is useful as an alternative to hex editing.

CREATE VARIABLE (Db2 V11)

A new CREATE VARIABLE statement that was introduced in Db2 V11 is fully supported by RDX. For example:

```
CREATE VARIABLE TEST_VAR VARCHAR(30) DEFAULT 'Define variable'
```

DECLARE CURSOR

RDX supports the DECLARE CURSOR statement by transparently issuing SQL PREPARE, DESCRIBE, and DESCRIBE INPUT statements. The output and input SQLDAs are filled in as a result of these statements. You can then issue OPEN, FETCH, and CLOSE statements to process the answer set. The following code fragments illustrate the DECLARE CURSOR processing and contrast it with the PREPARE-DESCRIBE processing.

Example 1: DECLARE CURSOR processing

```
rank              = 5
"EXECSQL DECLARE C2 CURSOR FOR",
 "SELECT COMPANY_NAME, COMPANY_RANK",
  "FROM RDX.DEMO",
   "WHERE COMPANY_RANK < :rank",
    "ORDER BY COMPANY_RANK ASC"

"EXECSQL OPEN C2"
DO WHILE SQLCODE  = 0
   "EXECSQL FETCH FROM C2 INTO :comp, :rank"
   IF sqlcode    \= 0 THEN LEAVE
   SAY 'company_name =' comp 'company_rank =' rank
END
"EXECSQL CLOSE C2"
```

Example 2: PREPARE-DESCRIBE processing

```
rank              = 5
stm               =,
 "SELECT COMPANY_NAME, COMPANY_RANK",
  "FROM RDX.DEMO",
   "WHERE COMPANY_RANK < :rank",
    "ORDER BY COMPANY_RANK ASC"

"EXECSQL PREPARE S3 FROM :stm"
"EXECSQL DESCRIBE S3"
"EXECSQL DESCRIBE INPUT S3"
"EXECSQL OPEN C3"
DO WHILE SQLCODE  = 0
   "EXECSQL FETCH FROM C3 INTO :comp, :rank"
   IF sqlcode    \= 0 THEN LEAVE
   SAY 'company_name =' comp 'company_rank =' rank
END
"EXECSQL CLOSE C3"
```

See the sample REXX program in the RDXSMP14 member of the CRAICRAI library for working examples of these two modes of processing.

DECLARE GLOBAL TEMPORARY TABLE

RDX fully supports the DECLARE GLOBAL TEMPORARY TABLE statement.

Example:

```
"EXECSQL DECLARE GLOBAL TEMPORARY TABLE SESSION.DEMO",
"(",
" COMPANY_NAME       CHAR    (20)                NOT NULL,",
" COMPANY_RANK       SMALLINT                    NOT NULL",
")",
"ON COMMIT PRESERVE ROWS"
```

DECLARE STATEMENT

RDX ignores any SQL DECLARE STATEMENT that you specify in a REXX program.

However, RDX internally implements a DECLARE STATEMENT for all types of statements that may be required in its implementation of REXX to Db2 eXtensions. RDX is not completely flexible regarding the usage of SQL statements. RDX defines statements with *fixed names* that are to be used for specific purposes. This behavior requires the programmer to use these names and *only* these statement names for a defined purpose.

Namely, the statements S1 through S200 are defined to correspond to the cursor names C1 through C200 exactly. For example, you cannot use statement S10 with the cursor named C5 -- instead you *must* use C10.

Furthermore, the statement names S1 through S99 are to be used only with a DECLARE CURSOR or a DECLARE REXXSTEM statement. As such, you are allowed to have up to 99 cursors opened simultaneously.

Statements S100 and cursor C100 are reserved for SELECT INTO statements. These statements may *never* be used explicitly in a REXX application.

Statement names S101 through S200 and the corresponding cursor names C101 through C200 are reserved for ALLOCATE CURSOR statements that process result sets returned by Db2 stored procedures.

DECLARE REXXSTEM (RDX)

The DECLARE REXXSTEM statement is a proprietary extension of RDX that provides the functionality of the DECLARE CURSOR, OPEN, FETCH, and CLOSE statements in a single, composite command. You can code a DECLARE REXXSTEM statement to simplify the retrieval of query results that are reasonably small (so as not to over-utilize memory). To allow you to navigate large answer sets (such as when you are browsing the entire content of a table), RDX provides the following control parameters for the DECLARE REXXSTEM service:

```
"CNTL START nnnnnn"
```

The CNTL START parameter directs RDX to skip nnnnnn-1 rows in the answer set before returning row number nnnnnn. This row becomes the first one returned to your program. RDX then fetches and returns the remaining rows of the query result *or* the number of rows that are defined by the LIMIT parameter, whichever is smaller. The default value for the starting row number is 1.

```
"CNTL LIMIT nnnnnn"
```

The CNTL LIMIT parameter establishes a limit on the number of answer set rows that RDX returns to your program. The LIMIT parameter can be used in conjunction with the START parameter. The default value for the LIMIT parameter is 999999.

DECLARE REXXSTEM Syntax

"EXECSQL DECLARE Cn REXXSTEM FOR" *full-select-into*

where:

- Cn - is a cursor name to be associated with a REXX stem, n = 1 – 99
- *full-select-into* - is any valid SQL *full-select* statement as defined by the *Db2 SQL Reference.* The statement may optionally specify an INTO clause. Doing so allows you to specify explicit binding between column values and REXX host variables and to identify into which REXX stemmed variables the column values of the query result should be returned.

  NOTE
  The DECLARE REXXSTEM service does not support multi-row FETCH. Rather, multi-row FETCH is supported only through an explicitly issued FETCH statement.

Specifying the INTO Clause

Within the *full-select-into* clause of a DECLARE REXXSTEM, the INTO clause can be specified as in one of the following cases.

The INTO Clause is Omitted

When the INTO clause is omitted, RDX creates variable names as shown in the following example:

```
"EXECSQL DECLARE C1 REXXSTEM FOR",
 "SELECT COMPANY_NAME, COMPANY_RANK",
  "FROM RDX.DEMO",
   "WHERE COMPANY_RANK < 5"
```

This SQL statement returns the following REXX stem variables:

- C1.ROWS - number of returned rows
- C1.i.SQLDATA.j - result set columns value variable
- C1.i.SQLIND.j - result set columns indicator variable

where:

- C1 - is the cursor name.
- i - is the column number, in the order it was specified on the column list of the SELECT statement or, by default, as it was defined on CREATE TABLE statement.
- j - = 1 to C1.ROWS is the row index

  NOTE
  These variable names are compatible with the variable names that are returned by DSNREXX (without the '.J' suffix).

Specify the INTO Clause as a List of Host Variables

Consider the following example:

```
INTO :hv1:iv1, :hv2:iv2, …
```

Where:

- hv1, hv2 - are names of main host variables
- iv1, iv2 - are names of indicator variables. An indicator variable must not be specified for any column that is defined with a NOT NULL clause.

```
"EXECSQL DECLARE C2 REXXSTEM FOR",
 "SELECT COMPANY_NAME, COMPANY_RANK",
```

```
  "INTO :company, :rank",
   "FROM RDX.DEMO",
    "WHERE COMPANY_RANK < 5"
```

This SQL statement returns the following REXX stem variables:

- C2.ROWS - is the number of rows returned
- company.j - values for the COMPANY_NAME column
- rank.j - values for the COMPANY_RANK column
- RDX implicitly creates indicator variables of the form:
- company_.j - indicator variable values for the COMPANY_NAME column
- rank_.j - indicator variable value for the COMPANY_RANK column

where:

- C2 - is the cursor name
- j - = 1 to C2.ROWS is a row index

Specify INTO SQLNAME

Usage of SQLNAME keyword results in the creation of a stem variable of the form:

Cn.columne_name.i. For example, the following statement:

```
"EXECSQL DECLARE C3 REXXSTEM FOR",
 "SELECT COMPANY_NAME, COMPANY_RANK",
  "INTO SQLNAME",
   "FROM RDX.DEMO",
    "WHERE COMPANY_RANK < 5"
```

In this format, the REXX variable names are the same as the SQL column names from the answer set. As such, the INTO SQLNAME specification returns variable names of the form:

- C3.ROWS - is the number of returned rows
- C3.company_name.j - values for the COMPANY_NAME column
- C3.company_rank.j - values for the COMPANY_RANK column
- C3.company_name_.j - indicator variable values for the COMPANY_NAME column
- C3.company_rank_.j - indicator variable values for the COMPANY_RANK column

where:

- C3 - is the cursor name
- j - = 1 to C3.ROWS is a row index

The sample REXX program in the RDXSMP13 member of the hlq.CRAIEXEC library is a working example and illustrates all three methods of INTO clause specification.

Specify INTO SQLNAMEO

The SQLNAMEO keyword was implemented to provide RDX compatibility with similar ISV products, such as MAX/REXX. The output stem variables have the format column_name.i. For example, the following SQL statement:

```
"EXECSQL DECLARE C2 REXXSTEM FOR",
  "SELECT LOCATION, LINKNAME, TRUSTED",
   "FROM SYSIBM.LOCATIONS",
    "INTO SQLNAMEO",
     "FETCH FIRST 2 ROWS ONLY"
```

produces output similar to the following example:

```
c2.rows.................. 2
location.1.............. RADSN
linkname.1.............. Db2DSN
trusted.1............... N
location.2.............. RADSNA
linkname.2.............. Db2DSNA
trusted.2............... N
```

DECLARE ISPFTABLE (RDX)

The DECLARE ISPFTABLE statement is a proprietary extension of RDX that provides the functionality of the DECLARE CURSOR, OPEN, FETCH, and CLOSE statements in a single, composite command. Fetched Db2 table rows are saved in the ISPF table, column names of which are the same as the Db2 column names if these names adhere to the ISPF table variable naming convention. If you specify an INTO clause, you can assign ISPF table column names to be any valid name, such as INTO :v1, :v2, … If you *did not specify* an INTO clause, ISPF table columns names are assigned using the following rules:

- If the Db2 table column name is not longer than eight characters and *does not contain* underscore characters, the corresponding ISPF column name is the same.
- If the Db2 table column name is not longer than eight characters but *does contain* one or more underscore characters, the corresponding ISPF table column name is the same as the Db2 column name with underscore characters that are substituted by the character that is defined in the **RDX$TSD** default module, field **RDXURC**. The default is '**#**' character. For example, if the Db2 table column name is COL_1_E, then the corresponding ISPF table name becomes COL#1#E.
- If the Db2 table column name length is greater than eight characters, the ISPF table column name is generated as PPPPnnn, where PPPP is a variable prefix that is defined in the **RDX$TSD** module by the field **RDXVPREF**. The default is 'RDXV'. You can change this prefix dynamically with the "CNTL PREFIX pppp" command. The nnn is the order of the column in the result set that is defined in the FROM clause.

You can code a DECLARE ISPFTABLE statement to simplify the retrieval of query results that are reasonably small (so as not to over-utilize memory). To limit the number of rows that are loaded in the ISPF table, use the following CNTL services that are described in the DECLARE REXXSTEM service:

```
"CNTL START nnnnnn"
"CNTL LIMIT nnnnnn"
```

DECLARE ISPFTABLE Syntax

```
"EXECSQL DECLARE" ispf_table_name "ISPFTABLE FOR" full-select-into
```

where:

ispf_table-name - is any valid ISPF table name

full-select-into - is any valid SQL *full-select* statement as defined by the *Db2 SQL Reference.* The statement may optionally specify an INTO clause which can specify the variable names to be used as the ISPF table column variables.

> **NOTE**
> The DECLARE ISPFTABLE service does not support multi-row FETCH. Rather, multi-row FETCH is supported only through an explicitly issued FETCH statement.

For example, the following statement:

```
name              = 'SYSV%'
"EXECSQL DECLARE SYSTABLE ISPFTABLE FOR",
  "SELECT NAME, DBNAME, ENCODING_SCHEME",
```

```
    "FROM SYSIBM.SYSTABLES",
       "WHERE NAME LIKE :name",
         "ORDER BY DBNAME, NAME"
```

produces the following output in ISPF table SYSTABLE:

```
               NAME             DBNAME            RDXV003
-------------------------------------------------------------
  SYSVARIABLES_DESC            DSNDB06                  U
       SYSVARIABLES            DSNDB06                  U
  SYSVARIABLES_TEXT            DSNDB06                  U
. . .
-------------------------------------------------------------
```

The following statement:

```
"EXECSQL DECLARE LOCATION ISPFTABLE FOR",
  "SELECT * FROM SYSIBM.LOCATION"
```

produces output in LOCATION ISPF table:

```
LOCATION LINKNAME  IBMREQD     PORT      TPN  DBALIAS  TRUSTED   SECURE
-------------------------------------------------------------------------------
   RADSN   Db2DSN        N                                  N        N
   RADSNA  Db2DSNA        N      472                         N        N
   DBGLID  Db2DBGA        N      470                         N        N
. . .
-------------------------------------------------------------------------------
```

DECLARE TABLE

RDX ignores any DECLARE TABLE statement.

DECLARE VARIABLE

RDX implements the DECLARE VARIABLE statement as a supporting structure for other SQL statements that require the explicit definition of a variable's data type and attributes (such as length and CCSID). You might need to change the data type of a column, as returned by Db2 through a DESCRIBE statement, to some other data type. For example, to retrieve or store XML columns, you must first change the data type to VARCHAR or LOB since Db2 does not support XML columns directly. You can manually construct the SQLDA from REXX stemmed variables. Doing so removes the need for declared variables but requires more programming effort and demands a thorough understanding of the dynamic SQL interface.

The following example describes the syntax of the DECLARE STATEMENT as implemented by RDX. Note that this syntax is *not compatible* with the syntax of the SQL DECLARE VARIABLE statement.

```
DECLARE varname VARIABLE
 DATATYPE(sqltype) {LENGTH(length) | LENGTH(precision,scale)} |
 datatype_name
   INPUT | OUTPUT
   INDICATOR
```

where:

- varname - is the name of the variable being declared. This variable is either the name of a column within the Db2 table or the name of a host variable (such as the SQL CALL statement and several other statements).
- sqltype - is the decimal value of the SQLTYPE field of the SQLDA, as documented in the *Db2 SQL Reference.* The following sample offers a brief summary:

```
SQLTYPE      Data type name
384          DATE
388          TIME
392          TIMESTAMP
400          NULL SEP GR STRING
404          BLOB
408          CLOB
412          DBCLOB
448          VARCHAR
452          CHAR
456          LONG VARCHAR
460          NULL CHARSTRING
464          VARGRAPHIC
468          GRAPHIC
472          LONG VARGRAPHIC
480          FLOAT
484          DECIMAL
492          BIG INTEGER
496          INTEGER
500          SMALLINT
504          DISPLAY SIGN
904          ROWID
908          VARBINARY
912          BINARY
916          BLOB FILE
920          CLOB FILE
924          DBCLOB FILE
960          BLOB LOCATOR
964          CLOB LOCATOR
968          DBCLOB LOCATOR
972          RESULT SET LOCATOR
976          TABLE LOCATOR
988          XML
996          DECFLOAT
```

Length - For all non-decimal data types, this variable specifies the length of the variable. Some data types (such as DATE and TIME) have an implicit length. When a data type with an implicit length is encountered, the LENGTH parameter is ignored.

precision - For a decimal data type, this variable defines the precision value.

scale - For a decimal data type, this variable defines the scale value

INPUT - Specifies that the declared variable is used as an input variable

OUTPUT - Specifies that the declared variable is used as an output variable. This value is the default.

INDICATOR - Specifies that the declared variable is to be used as an indicator variable. The varibale must have a data type of SMALLINT.

```
CCSID {EBCDIC|ASCII|UNICODE}
```

```
        FOR {SBCS|MIXED|BIT} DATA
```

where

- CCSID - is an integer, where an integer denotes a valid CCSID to be used with the variable.

The following datatype_name specification is an acceptable alternative to specifying the DATATYPE(sqltype) parameter:

```
SMALLINT
INTEGER | INT
BIGINT
DECIMAL | DEC | NUMERIC (precision,scale)
FLOAT (53) | (int)
REAL
DOUBLE {PRECISION}
DECFLOAT (34) | (16)
CHARACTER | CHAR  (1) | (int)
(CHARACTER | CHAR) VARYING  | VARCHAR  (int)
CHAR LARGE OBJECT | CLOB  (1M) | (int K|M|G)
GRAPHIC (int) | (1)
VARGRAPHIC (int)
DBCLOB (1M) | (int K|M|G)
BINARY (1) | (int)
BINARY VARYING | VARBINARY (int)
BINARY LARGE OBJECT (1M) | BLOB (int K|M|G)
DATE
TIME
TIMESTAMP
ROWID
(CLOB | DBCLOB | BLOB ) LOCATOR
(CLOB | DBCLOB | BLOB ) FILE
```

Example:

The following examples are *equivalent* definitions.

```
"EXECSQL DECLARE column_name VARIABLE VARCHAR(254) INPUT"
```

and

```
“EXECSQL DECLARE column_name DATATYPE(448) LENGTH(254) INPUT”
```

See the sample REXX program in the RDXSMP02 member of the CRAIEXEC library for a working example.

DELETE

The DELETE statement is fully supported by RDX and is implemented by one of the following methods:

- When the DELETE statement contains no embedded host variables, an EXECUTE IMMEDIATE is used.
- When the DELETE statement contains embedded host variables, RDX implements the DELETE request through PREPARE, DESCRIBE, and EXECUTE USING SQLDA statements.

Example 1: A DELETE statement with embedded host variables (taken from the sample program RDXSMP11)

```
rank = 8888
"EXECSQL DELETE FROM RDX.DEMO",
  "WHERE COMPANY_RANK = :rank"
SAY 'Deleted' sqlerrd.3 'rows'
```

See the related examples in the descriptions for the INSERT and UPDATE statements.

Example 2: A DELETE statement without embedded host variables

```
Stm = "DELETE FROM RDX.DEMO",
  "WHERE COMPANY_RANK = 8888"
"EXECSQL EXECUTE IMMEDIATE :stm"
SAY 'Deleted' sqlerrd.3 'rows'
```

This example was taken from the sample program RDXSMP12.

DESCRIBE CURSOR

RDX supports the DESCRIBE CURSOR statement using the following syntax:

```
"EXECSQL DESCRIBE CURSOR :csrname INTO :sqlda"
```

where:

- csrname - is a host variable containing the name of the cursor to be described
- sqlda - is a REXX stem into which the SQLDA describing the cursor is externalized

Example:

```
csr  = 'C102'
"EXECSQL DESCRIBE CURSOR :csr INTO :sda"
```

In this example, we assume that the cursor C102 was opened through an ALLOCATE statement because it is in the range 101 through 200.

See the sample REXX program in the RDXSMP05 member of the CRAIEXEC library for a working example of the DECLARE CURSOR statement.

DESCRIBE INPUT

RDX supports the DESCRIBE INPUT statement using the following syntax:

```
EXECSQL DESCRIBE INPUT Sn INTO :sqlda
```

where:

- Sn - is a statement name, n = 1 through 99
- sqlda - is a REXX stem into which the SQLDA describing the statement is externalized

Example:

```
stm = 'SELECT * FROM SYSIBM.SYSSTOGROUP WHERE NAME = :name'
"EXECSQL PREPARE S2 FROM :stm USING BOTH"
"EXECSQL DESCRIBE INPUT S2 INTO :in"
```

In this example, the SQL statement is assigned to the variable stm. The statement is first prepared, then described into the stem named 'in.'. The SQLDA with the stem name 'in.' contains a description of one variable – name as shown in the following example:

```
IN.SQLDAID..................... = SQLDA
IN.SQLD........................ = 1
---- Column 1 -----------------------------
IN.1.SQLTYPE................... = 449
IN.1.SQLLEN.................... = 128
IN.1.SQLCCSID.................. = 1208
```

```
IN.1.SQLNAME.................. = NAME
IN.1.SQLLABEL................. = NAME
IN.1.SQLDTSN.................. = VC
IN.1.SQLDTLN.................. = VARCHAR
```

See program RDXSMP10 for a working example.

DESCRIBE OUTPUT

RDX supports DESCRIBE statements using the following syntax:

```
EXECSQL DESCRIBE Sn
EXECSQL DESCRIBE Sn INTO :sqlda
EXECSQL DESCRIBE Sn INTO :sqlda USING BOTH
EXECSQL DESCRIBE Sn INTO :sqlda USING ANY
EXECSQL DESCRIBE Sn INTO :sqlda USING LABELS
```

where:

- Sn - is a statement name, n = 1 through 99
- sqlda - is a REXX stem into which the SQLDA describing the statement is externalized

> **NOTE**
> DESCRIBE Sn (without :sqlda) means that there is no REXX stem into which the SQLDA is externalized. However, RDX creates an internal SQLDA so the statement can be EXECUTE-ed.

See the sample REXX program in the RDXSMP10 member of the CRAIEXEC library for a working example.

DESCRIBE PROCEDURE

RDX supports the DESCRIBE PROCEDURE statement using the following syntax:

```
EXECSQL DESCRIBE PROCEDURE :procname INTO :sqlda
```

where:

- procname - is a variable containing the name of the stored procedure to be described
- sqlda - is a REXX stem into which the SQLDA describing the procedure is externalized

> **NOTE**
> Before you issue a DESCRIBE PROCEDURE statement, the currently active application thread must first successfully CALL the procedure.

See the sample REXX program in the RDXSMP05 member of the CRAIEXEC library for a working example.

DESCRIBE TABLE

RDX supports all forms of the DESCRIBE TABLE statement as shown in the following code sample:

```
EXECSQL DESCRIBE TABLE :table INTO :sqlda USING BOTH
EXECSQL DESCRIBE TABLE :table INTO :sqlda USING ANY
EXECSQL DESCRIBE TABLE :table INTO :sqlda USING LABELS
EXECSQL DESCRIBE TABLE :table INTO :sqlda
```

where:

- table - is a variable containing the name of the table to be described
- sqlda - is a REXX stem into which the SQLDA describing the table is externalized

See the sample REXX program in the RDXSMP10 member of the CRAIEXEC library for a working example.

DROP

RDX fully supports the DROP statement. DROP may be issued directly (as illustrated in the example) or through an EXECUTE IMMEDIATE statement (as discussed in the documentation for the ALTER statement).

Example:

```
"EXECSQL DROP TABLE RDX.DEMO"
```

END DECLARE SECTION

RDX does not support the END DECLARE SECTION statement.

EXCHANGE (Db2 V9)

RDX fully supports the EXCHANGE statement. EXCHANGE may be issued directly or through an EXECUTE IMMEDIATE statement (as discussed in the documentation for the ALTER statement).

To use the EXCHANGE statement, the source table must first be defined in a Universal Table Space (UTS). In addition, a clone table must be created for the source table. Only then can an EXCHANGE statement be executed successfully.

See the sample REXX program in the RDXSMP12 member of the CRAIEXEC library for a working example.

EXECUTE

RDX supports the EXECUTE statement using the following syntax:

```
EXECUTE S1
EXECUTE S1 USING DESCRIPTOR :stem
EXECUTE IMMEDIATE :statement
```

where:

- stem - is the name of a REXX stem in which the SQLDA structure is stored.
- statement - is a REXX variable that contains a valid non-SELECT SQL statement to be executed.

All the following examples appear in the RDXSMP15 program.

Example 1: EXECUTE a prepared statement

Suppose we want to execute the following SQL statement:

```
statement =,
"UPDATE RDX.DEMO",
  "SET   COMPANY_NAME =:comp,",
        "COMPANY_RANK = :rank ",
  "WHERE COMPANY_RANK = :rank"
comp              = 'Company' rank 'updated'
rank              = 25
"EXECSQL PREPARE S1 FROM :statement"
```

In Example 1 above, the SQL statement text with embedded host variables is assigned to the variable named ‘statement’. Note that host variables are denoted in the SQL statement text as variable names preceded by a colon ‘:’ (as in :comp and:rank) rather than by the question mark character ‘?’ used to denote a parameter marker in dynamic SQL syntax. Using host variable names allows RDX to bind variable values to an internally constructed SQLDA structure. Values are assigned to each input host variable (in this example :comp and :rank) *before* the statement is PREPAREd. Thus, the

internal SQLDA can point to assigned variable values. Next, a DESCRIBE INPUT request is issued for the prepared statement as in:

```
"EXECSQL DESCRIBE INPUT S1"
```

The statement can then be executed, as in:

```
"EXECSQL EXECUTE S1"
```

Example 2: EXECUTE USING DESCRIPTOR

Use the same assignment and PREPARE statements from Example 1 and then issue a DESCRIBE INTO request as follows:

```
statement =,
"UPDATE RDX.DEMO",
  "SET   COMPANY_NAME =:comp,",
        "COMPANY_RANK = :rank ",
  "WHERE COMPANY_RANK = :rank"
comp              = 'Company' rank 'updated'
rank              = 25
"EXECSQL PREPARE S1 FROM :statement"
"EXECSQL DESCRIBE INPUT S1 INTO : id "
```

This DESCRIBE INPUT request describes statement S1 into an input SQLDA that RDX externalizes to the REXX stem named ‘id.’ Alternatively, you can manually build the desired SQLDA yourself by assigning values to stem variables. Once described, statement S1 can be executed using the descriptor that is mapped by REXX variables that share the stem id.

```
"EXECSQL EXECUTE S1 USING DESCRIPTOR :id"
```

Example 3: EXECUTE IMMEDIATE

When the SQL statement contains no embedded host variables, processing (and coding) can be abbreviated. Instead of issuing PREPARE, DESCRIBE INPUT and EXECUTE requests in sequence, simply issue an EXECUTE IMMEDIATE statement as in:

```
statement =,
"UPDATE RDX.DEMO",
  "SET   COMPANY_NAME = ‘Company 25 updated’",
        "COMPANY_RANK = 25",
  "WHERE COMPANY_RANK = 25"
"EXECSQL EXECUTE IMMEDIATE :statement"
```

The code in Example 3 is simpler than the code in Examples 1 and 2. However, the EXECUTE IMMEDIATE syntax lacks generality and does not work with data types whose values cannot be coded as literals within the SQL statement, but must instead be assigned to host variables.

EXPLAIN

RDX fully supports the EXPLAIN statement. See the sample stored procedure named RDXSPEXP within the CRAIEXEC library. RDXSPEXP is similar to the IBM supplied C language program named DSN8EXP. The RDXSPEXP stored procedure is defined as follows:

```
CREATE PROCEDURE
  RDX.RDXSPEXP
    (
    IN    SQLID             CHAR(8)                 CCSID EBCDIC,
    IN    QUERYNO           INTEGER
```

```
    IN    SQLTXT             VARCHAR(32700)          CCSID EBCDIC,
    OUT   ERRORMSG           VARCHAR(960)            CCSID EBCDIC
    )
    LANGUAGE REXX
    EXTERNAL NAME RDXSPEXP
    COLLID RDX110
    PARAMETER STYLE GENERAL
    STAY RESIDENT NO
    ASUTIME NO LIMIT
    WLM ENVIRONMENT DB9AWLM1
    PROGRAM TYPE MAIN
    SECURITY Db2
    COMMIT ON RETURN NO
    INHERIT SPECIAL REGISTERS;
```

The REXX code which comprises the stored procedure that is named RDXSPEXP can be distilled to the following statements:

```
/* REXX */
PARSE ARG sqlid,queryno,sqltxt
"EXECSQL SET CURRENT SQLID = :sqlid"
"EXECSQL EXPLAIN ALL SET QUERYNO =" queryno "FOR" sqltxt"
RETURN 'SQLCODE =' sqlcode 'SQLSTATE =' sqlstate
```

The RDXSPEXP procedure can be invoked with the following CALL statement:

```
sqlid = 'DSN10'
queryno = 5
sqltxt = 'SELECT * FROM RDX.DEMO'
"EXECSQL CALL RDX.RDXSPEXP (:sqlid, :queryno, :sqltxt)
```

Once implemented, the RDXSPEXP procedure provides a generic way to EXPLAIN SQL statements, as long as EXPLAIN tables are created under the SQLID DSN10 (used for the IBM-supplied samples) and users can execute with this SQLID.

FETCH

RDX fully supports all varieties of the FETCH statement that are documented in the *Db2 for z/OS SQL Reference,* including both single and multi-row fetch.

Example: Multiple row fetch

See the sample REXX program in the RDXSMP16 member of the CRAIEXEC library for a working example.

```
ADDRESS RDX
 CALL SDWCLEAR                        /* clear the screen */

/* REXX */
 ssid              = 'DB9A'           /* db2 subsystem name */
 "INIT SYSTEM("ssid")"
 CALL ShowErrorMsg 'INIT'

 rs#               = 100              /* size of result set */
 "EXECSQL DECLARE C1",           /* DECLARE CURSOR with ROWSET */
    "INSENSITIVE SCROLL",
  "CURSOR",
   "WITHOUT HOLD",
```

```
    "WITHOUT RETURN",
     "WITH ROWSET POSITIONING",
  "FOR",
  "SELECT COMPANY_NAME, COMPANY_RANK",
   "FROM RDX.DEMO",
    "WHERE COMPANY_RANK ' :rs#",
     "ORDER BY COMPANY_RANK ASC"
 CALL ShowErrorMsg 'DECLARE'

 "EXECSQL OPEN C1"
 CALL ShowErrorMsg 'OPEN'

 rowset            = 'ROWSET STARTING AT ABSOLUTE :pos'
 for_rows          = 'FOR :rows ROWS'
into              = 'INTO :comp,:rank'
 rows              = 2                 /* by 2 rows          */
 pos               = 3                 /* start from row 3   */

 SAY '>> Rows 3,4'
 "EXECSQL FETCH" rowset "FROM C1" for_rows into
 CALL ShowErrorMsg 'FETCH'
 SAY '>> Fetched rows='sqlerrd.3
 CALL ShowFetched

 pos               = 51                /* start from row 51  */
 rows              = 3                 /* by 3 rows          */
 SAY 'Rows 51,52,53'
 "EXECSQL FETCH" rowset "FROM C1" for_rows into
 CALL ShowErrorMsg 'FETCH'
 SAY '>> Fetched rows='sqlerrd.3
 CALL ShowFetched

 pos               = 99                /* start from row 51  */
 rows              = 3                 /* by 3 rows          */
 SAY 'Row 99 and SQLCODE = 100'
 "EXECSQL FETCH" rowset "FROM C1" for_rows into
 CALL ShowErrorMsg 'FETCH'
 SAY '>> Fetched rows='sqlerrd.3
 CALL ShowFetched

 "EXECSQL CLOSE C1"
 CALL ShowErrorMsg 'CLOSE'

 "TERM"
 CALL ShowErrorMsg 'TERM'
RETURN 0
/*_____________________________________________________________*/
/* Display fetched row(s)                                      */
/*_____________________________________________________________*/
ShowFetched :
 DO i              = 1 TO sqlerrd.3
    SAY '> company =' comp.i 'rank =' rank.i
 END
```

```
 CALL Diag
RETURN 0
/*_____________________________________________________________*/
/* Get diagnostic information                                  */
/*_____________________________________________________________*/
Diag :
 SAY 'GET DIAGNOSTICS'
 "EXECSQL GET DIAGNOSTICS :stmt = ALL STATEMENT"
 CALL UnpackVars 'STATEMENT','stmt'
RETURN 0
/*_____________________________________________________________*/
/* Unpack variables returned by GET DIAGNOSTICS                */
/*_____________________________________________________________*/
UnpackVars :
ARG type,var
 value             = VALUE(var)
 SAY '***' type '***'
 DO WHILE value \= ''
    PARSE VAR value name'='val';'value
    SAY LEFT(name,36,'.') val
 END
RETURN 0
/*_____________________________________________________________*/
/* Issue error messages                                        */
/*_____________________________________________________________*/
ShowErrorMsg :
TRACE 'o'
 ARG stn
 SAY LEFT(stn,12) '- SQLCODE =' sqlcode 'RC =' rc 'REASON ='
reason
 IF sqlcode        = 0 THEN RETURN 0
 DO i              = 1 TO SQLERR.0
    SAY SQLERR.i
 END
 IF stn            = 'INIT' THEN EXIT 8
RETURN 0
```

An output trace from the execution of this program appears as follows:

```
INIT - SQLCODE = 0 RC = 0 REASON = 00000000
DECLARE - SQLCODE = 0 RC = 0 REASON = 00000000
OPEN - SQLCODE = 0 RC = 0 REASON = 00000000
>> Rows 3,4
FETCH - SQLCODE = 0 RC = 0 REASON = 00000000
>> Fetched rows=2
> company = Company 0003 rank = 3
> company = Company 0004 rank = 4
GET DIAGNOSTICS
*** STATEMENT ***
ROW_COUNT.......................... 02
MORE............................... N
Rows 51,52,53
FETCH - SQLCODE = 0 RC = 0 REASON = 00000000
>> Fetched rows=3
```

```
> company = Company 0051 rank = 51
> company = Company 0052 rank = 52
> company = Company 0053 rank = 53
GET DIAGNOSTICS
*** STATEMENT ***
ROW_COUNT..... 03
MORE..... N
Row 99 and SQLCODE = 100
FETCH - SQLCODE = 100 RC = 1 REASON = 00000000
DSNT404I SQLCODE = 100, NOT FOUND: ROW NOT FOUND FOR FETCH,
UPDATE, OR
 DELETE, OR THE RESULT OF A QUERY IS AN EMPTY TABLE
DSNT418I SQLSTATE = 02000 SQLSTATE RETURN CODE
DSNT415I SQLERRP = DSNXRFN SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD = 0 99 1 -1 0 0 SQL DIAGNOSTIC
INFORMATION
DSNT416I SQLERRD = X'00000000' X'00000063' X'00000001'
X'FFFFFFFF'
 X'00000000' X'00000000' SQL DIAGNOSTIC INFORMATION
>> Fetched rows=1
> company = Company 0099 rank = 99
GET DIAGNOSTICS
*** STATEMENT ***
Db2_LAST_ROW..... 100
NUMBER..... 1
ROW_COUNT..... 01
Db2_NUMBER_ROWS..... 099
MORE..... N
CLOSE - SQLCODE = 0 RC = 0 REASON = 00000000
   TERM - SQLCODE = 0 RC = 0 REASON = 00000000
```

FREE LOCATOR

RDX fully supports the FREE LOCATOR statement using the following syntax:

```
"EXECSQL FREE LOCATOR :locator1, :locator, . . ."
```

Example: Usage of HOLD and FREE LOCATOR

In the following example, the REXX host variable that is named SRCE_LOC is first declared as a CLOB LOCATOR. Next, the CLOB column that is named SRCE is selected into SRCE_LOC. Third, the locator value that is stored in the variable srce_loc becomes the target of a HOLD statement, such that the locator value remains valid after the COMMIT statement which follows. Eventually, the locator is FREE-ed so it can be reused.

```
"EXECSQL DECLARE SRCE_LOC VARIABLE CLOB LOCATOR"
name              = 'PROG001'
"EXECSQL SELECT SRCE INTO :SRCE_LOC",
 "FROM RDX.TESTLOBS",
  "WHERE NAME = :name"

"EXECSQL HOLD LOCATOR :srce_loc"

. . . some processing . . .
```

```
"EXECSQL COMMIT"
"EXECSQL FREE LOCATOR :srce_loc"
```

Refer to the sample REXX program in the RDXSMP18 member of the CRAIEXEC library for a working example.

GET DIAGNOSTICS (Db2 V8)

RDX implements the GET DIAGNOSTICS statement that was introduced in Db2 Version 8 with the following abbreviated syntax:

```
GET DIAGNOSTICS :host-variable =
     ALL STATEMENT |
     ALL CONDITION |
     ALL CONNECTION
```

This syntax is a form of *combined-information* syntax that is supported by the SQL GET DIAGNOSTICS statement. The syntax provides a text representation of all the information that is gathered about the execution of the SQL statement. ALL indicates that all diagnostic items that are associated with the last executed SQL statement are to be combined into one string. The format of the string that is returned by GET DIAGNOSTICS is a list of all the available diagnostic information, which is delimited by semicolons, in the form:

```
<item-name>[(<condition-number>)]=<value-converted-to-character>;
...
```

The following example illustrates a GET DIAGNOSTICS string:

```
NUMBER=1;RETURNED_SQLSTATE=02000;Db2_RETURNED_SQLCODE=+100;
```

Despite the abbreviated syntax, all conditions that are defined by the syntax of the SQL GET DIAGNOSTICS statement can be obtained by extracting the sought condition from the output *host-variable* to which the diagnostic string is assigned.

Example: Issue a GET DIAGNOSTICS request and extract the condition information from the output host variable.

This example uses each GET DIAGNOSTICS statement that RDX s supports. After the GET DIAGNOSTICS statement is issued, a call is made to a REXX subroutine that unpacks and displays the conditions that are returned by Db2.

```
CALL EXECSQL "GET DIAGNOSTICS :stmt = ALL STATEMENT"
 CALL UnpackVars 'STATEMENT','stmt'
CALL EXECSQL "GET DIAGNOSTICS :cond = ALL CONDITION"
 CALL UnpackVars 'CONDITION','cond'
CALL EXECSQL "GET DIAGNOSTICS :conn = ALL CONNECTION"
 CALL UnpackVars 'CONNECTION','conn'

UnpackVars :
ARG type,var
 IF sqlcode       \= 0 THEN RETURN 8
 value             = VALUE(var)
 SAY '***' type '***'
 DO WHILE value \= ''
    PARSE VAR value name'='val';'value
    SAY LEFT(name,36,'.') val
 END
RETURN 0
```

GRANT Statements

RDX fully supports the GRANT statement. GRANT may be issued directly (as illustrated in the example) or through an EXECUTE IMMEDIATE statement (as discussed in the documentation for the ALTER statement).

Example:

```
"EXECSQL GRANT SYSADM TO IBMUSER"
```

HOLD LOCATOR

RDX fully supports the HOLD LOCATOR statement. See the documentation for the FREE LOCATOR statement for an example and further discussion.

INCLUDE

RDX does not support the INCLUDE statement.

INSERT

The INSERT statement is fully supported by RDX.

Example: Insert table rows

The following example appears in the sample program that is named RDXSMP11, which is supplied as a member of the CRAIEXEC library.

```
rank = 8888
hv1            = 'New company'
hv2            = 9999
hv3            = 'John Smith'
hv4            = '2007-03-12'
hv5            = '12.35.46'
hv6            = '2007-03-12-12.35.56.938759'
hv7            = 100
hv8            = 200
hv9            = 25
hv10           = 300
hv11           = 5000.00
hv12           = 'A11'
hv13           = 'This company is inserted'
"EXECSQL INSERT INTO RDX.DEMO",
 "VALUES(:hv1, :hv2, :hv3, :hv4, :hv5, :hv6, :hv7,",
        ":hv8, :hv9, :hv10,:hv11,:hv12,:hv13)"
CALL ShowErrorMsg 'INSERT'
SAY 'Inserted' sqlerrd.3 'rows'
```

For more information, see the documentation and examples for the DELETE and UPDATE statements.

LABEL

RDX fully supports the LABEL statement. LABEL may be issued directly (as illustrated in the example) or through an EXECUTE IMMEDIATE statement (as discussed in the documentation for the ALTER statement).

Example:

```
"EXECSQL LABEL ON COLUMN RDX.DEMO.COMPANY_NAME IS 'Company'"
```

LOCK TABLE

RDX fully supports the LOCK TABLE statement. LOCK TABLE may be issued directly (as illustrated in the example) or through an EXECUTE IMMEDIATE statement (as discussed in the documentation for the ALTER statement).

Example:

```
"EXECSQL LOCK TABLE RDX.DEMO IN EXCLUSIVE MODE"
```

MERGE (Db2 V9)

RDX supports the MERGE statement in single row mode, as illustrated in the following example. RDX does *not* support the 'FOR :hv ROWS' clause.

Example:

```
"EXECSQL MERGE INTO RDX.DEMOMERGE2 AR",
 "USING (VALUES (:comp, :rank))",
  "AS AC (COMPANY_NAME, COMPANY_RANK)",
 "ON (AR.COMPANY_RANK = AC.COMPANY_NAME)",
 "WHEN MATCHED THEN UPDATE SET COMPANY_NAME = AC.COMPANY_NAME",
 "WHEN NOT MATCHED THEN INSERT (COMPANY_NAME, COMPANY_RANK)",
      "VALUES(AC.COMPANY_NAME, AC.COMPANY_RANK)",
 "NOT ATOMIC CONTINUE ON SQLEXCEPTION"
```

OPEN

RDX supports two forms of the OPEN statement:

```
"EXECSQL OPEN Cn"
"EXECSQL OPEN Cn USING DESCRIPTOR :stem"
```

where:

- Cn - is a cursor name n = 1 through 99
- stem - is the name of REXX stem from which RDX builds an internal SQLDA

Example 1: In the following example, the cursor C5 (associated with the SQL statement named S5) is opened. If the SQL statement prepared as S5 contains input host variables and a DESCRIBE INPUT request for statement S5 is not explicitly coded, then RDX internally executes DESCRIBE INPUT for statement S5.

```
"EXECSQL OPEN C5"
```

Example 2: In the following example, the cursor named C20 (associated with the SQL statement S20) was first prepared (not shown). Then, the cursor's input host variables are described using the DESCRIBE INPUT S20 statement. Finally, an OPEN C20 request is issued which specifies the name of a REXX stem isda which supplies the values to be mapped into the input SQLDA. RDX constructs an internal SQLDA structure from these REXX stemmed variables and then executes OPEN using this SQLDA.

```
"EXECSQL DESCRIBE INPUT S20 INTO : isda"
"EXECSQL OPEN C20 USING DESCRIPTOR :isda"
```

> **NOTE**
> If you prepared a SELECT statement that uses input host variables in the WHERE clause, you can issue explicitly the DESCRIBE INPUT statement. Or, if you did not, the OPEN processing automatically issues the DESCRIBE INPUT when required.

PREPARE

RDX fully supports the SQL PREPARE statement using the following syntax:

```
PREPARE Sn FROM :statement
     { INTO :sqlda }
     { ATTRIBUTES :attt:attriv }
     { USING BOTH | ANY | LABELS }
```

where:

- statement - is a host variable containing the SQL statement to be prepared
- sqlda - is the name of a REXX stem into which RDX stores the SQLDA of the described statement
- attr - is a host variable containing a list of valid attributes, such as ‘HOLD SENSITIVE’ and so on. For more information, see the Db2 SQL Reference.
- attriv - is an indicator variable for the attr variable. You can set this indicator variable to -1 (denoting the null value) to disable the list of attributes specified with the attr variable.

  NOTE
  If the USING clause of the PREPARE statement is specified, then the INTO clause must also be specified.

See the sample REXX program in the RDXSMP10 member of the CRAIEXEC library for a working example of SQL PREPARE statements.

REFRESH TABLE

RDX fully supports the REFRESH TABLE statement. REFRESH TABLE may be issued directly (as illustrated in the example) or through an EXECUTE IMMEDIATE statement (as discussed in the documentation for the ALTER statement).

Example:

```
“EXECSQL REFRESH TABLE MQT1”
```

RELEASE Connection

RDX fully supports all forms of the RELEASE statement.

```
“EXECSQL RELEASE :server”
“EXECSQL RELEASE CURRENT”
“EXECSQL RELEASE ALL”
“EXECSQL RELEASE ALL SQL”
“EXECSQL RELEASE ALL PRIVATE”
```

- **server** - is the name of the remote server whose connection is to be released

RELEASE SAVEPOINT

RDX supports the RELEASE SAVEPOINT statement by issuing an EXECUTE IMMEDIATE statement internally.

Program RDXSMP09 illustrates the RELEASE SAVEPOINT statement.

RENAME

RDX fully supports the RENAME statement. RENAME may be issued directly (as illustrated in the example) or through an EXECUTE IMMEDIATE statement (as discussed in the documentation for the ALTER statement).

Example:

```
"EXECSQL RENAME DBX.DEMO RDX.DEMO"
```

REVOKE Statements

RDX fully supports the SQL REVOKE statements. REVOKE may be issued directly or through an EXECUTE IMMEDIATE statement (as discussed in the documentation for the ALTER statement).

ROLLBACK

RDX supports the SQL ROLLBACK statement and the CPIC RRSBACK function. The syntax of the RDX ROLLBACK statement is as follows:

```
ROLLBACK {WORK} TO SAVEPOINT savepoint-name
         {CPIC}
```

When the RRSAF attachment is used instead of CAF, the CPIC RRSBACK function can be requested by issuing the following statement:

```
ROLLBACK CPIC
```

The discussion of the COMMIT statement also applies to the RRSBACK function.

If the SAVEPOINT keyword is coded, then RDX executes the ROLLBACK request through an EXECUTE IMMEDIATE statement.

Program RDXSMP09 illustrates various forms of ROLLBACK statements.

SAVEPOINT

RDX supports the SAVEPOINT statement by internally issuing an EXECUTE IMMEDIATE request. Program RDXSMP09 illustrates various forms of SAVEPOINT statements.

SELECT INTO

RDX supports the SELECT INTO statement by internally issuing DECLARE CURSOR, OPEN, FETCH, and CLOSE statements. Therefore, you can view SELECT INTO as shorthand for cursor processing. Two cases of SELECT INTO are described:

- The SELECT INTO statement specifies a FETCH FIRST 1 ROWS ONLY clause. In this case, only the first row is fetched. If this row exists, then the SQLCODE is set to 0.
- No FETCH FIRST 1 ROWS ONLY clause is specified. In this case, two FETCHs are performed. If the second FETCH succeeds (that is, the answer set contains more than one row), then RDX sets the SQLCODE to -811 to simulate the behavior of the static SELECT INTO statement.

The RDX implementation of the SELECT INTO statement behaves the same way as its static SQL counterpart.

SET Statements

RDX directly supports most of the SET statements in the following list:

```
SET CONNECTION :server
SET CURRENT DEGREE = :degree
SET CURRENT PACKAGESET = USER
SET CURRENT PACKAGESET = :packageset
SET CURRENT SQLID = USER
SET CURRENT SQLID = :sqlid
SET CURRENT RULES = :rules
```

```
SET CURRENT PRECISION = :precision
SET CURRENT PATH = :path
SET CURRENT OPTIMIZATION HINT = :hint
SET CURRENT LOCALE LC_CTYPE = :locale
SET :SQLID = USER
SET :SQLID = CURRENT SQLID
SET :DATE = CURRENT DATE
SET :TIME = CURRENT TIME
SET :TS = CURRENT TIMESTAMP
SET :TMZN = CURRENT TIMEZONE
SET :PKSET = CURRENT PACKAGESET
SET :servrr = CURRENT SERVER
SET :DEGRE = CURRENT DEGREE
SET :RULES = CURRENT RULES
SET :PRCSN = CURRENT PRECISION
SET :PATH = CURRENT PATH
SET :HINT = CURRENT OPTIMIZATION HINT
SET :LOCAL = CURRENT LOCALE LC_CTYPE
```

Alternatively, a SELECT INTO statement can be used to obtain the value of a Db2 special register. The following example obtains the value of the CURRENT PRECISION special register by selecting into the REXX host variable named precision:

```
"EXECSQL SELECT CURRENT PRECISION INTO : precision FROM
     SYSIBM.SYSDUMMY1"
```

The SELECT INTO technique can be used to obtain the value of any Db2 special register that is not directly supported by a RDX SET statement. The sample REXX program in the RDXSMP01 member of the CRAIEXEC library provides a working example.

SET Statements for LOB LOCATOR Processing

When using LOB locators in conjunction with dynamic SQL, many SET statements that exploit the SQL scalar functions LENGTH, POSSTR, SUBSTR, and CONCAT cannot be dynamically prepared. For these cases, RDX implements a group of static SQL statements with fixed data types to support the processing scenarios for LOB locators. The following paragraphs describe and illustrate these static SQL SET statements. All parameters must be specified as host variables, as illustrated in the following examples. For illustrative purposes, the host variable names that are used in the statements match their data types. You can use any variable names in your own programs. Note that all locators in these examples are CLOB locators.

The following statement uses the LENGTH scalar function to return in the host variable named INT1, the length (as an integer value) of the CLOB identified by the CLOB locator CLOB_LOCATOR1.

```
SET :INT1 = LENGTH(:CLOB_LOCATOR1)
```

Example: Select a CLOB column into a CLOB LOCATOR and display its length.

```
"EXECSQL DECLARE SRCE     VARIABLE CLOB LOCATOR"
"EXECSQL DECLARE LOCATOR1 VARIABLE CLOB LOCATOR"
Name = 'PROG001'
"EXECSQL SELECT SRCE INTO :LOCATOR1 FROM RDX.TESTLOBS",
 "WHERE NAME = :name"
"EXECSQL SET :length = LENGTH(:LOCATOR1)"
SAY 'CLOB COLUMN LENGTH='length
```

The following statement uses the POSSTR scalar function to search a CLOB represented by the CLOB LOCATOR stored in the REXX host variable CLOB_LOCATOR1 for an occurrence of the string that is stored in the second parameter. This

second REXX host variable must be defined as VARCHAR(256). If the string is found, its starting position is returned as an integer value in the REXX host variable INT1. If not, a zero value is returned.

```
SET :INT1 = POSSTR(:CLOB_LOCATOR1,:VARCHAR1)
```

Example: Search for a string in a CLOB referenced by the CLOB LOCATOR variable named LOCATOR1. Return the starting position of the string as an integer value in the REXX host variable named ‘start’.

```
"EXECSQL DECLARE LOCATOR1 VARIABLE CLOB LOCATOR"
string            = 'PROG001 LINE 2'
"EXECSQL SET :start = POSSTR(:locator1, :string)"
SAY 'STRING START POSITION='start
```

The following statement uses the SUBSTR scalar function to create a CLOB locator that references a substring of the CLOB represented by the REXX host variable named CLOB_LOCATOR1. The substring starts at the relative character position that is stored in the REXX host variable INT1 and has the integer length that is stored in the REXX host variable INT2. Note that the content of a CLOB is not fetched into the caller’s virtual storage. Rather, the entire operation takes place in the Db2 DBM1 address space.

```
SET :CLOB_LOCATOR2 = SUBSTR(:CLOB_LOCATOR1,:INT1,:INT2)
```

Example: Create a CLOB locator that is named LOCATOR2 and assign to the CLOB it represents the substring value of the CLOB pointed to by the REXX host variable named LOCATOR1. The substringed CLOB to which the REXX host variable LOCATOR2 points, starts at the integer character position that is specified in the beg1 variable, with a length value specified by the variable end1.

```
"EXECSQL DECLARE LOCATOR1 VARIABLE CLOB LOCATOR"
"EXECSQL DECLARE LOCATOR2 VARIABLE CLOB LOCATOR"

beg1 = 1
end1 = 80
"EXECSQL SET :locator2 = SUBSTR(:locator1,:beg1,:end1)"
SAY 'SUBSTR('beg1','end1')'
```

The following statement uses the CONCAT scalar function to create a CLOB locator variable that is named CLOB_LOCATOR3 which represents a CLOB formed by concatenating the CLOB represented by CLOB_LOCATOR1 with the CLOB represented by CLOB_LOCATOR2.

```
SET :CLOB_LOCATOR3 = CONCAT(:CLOB_LOCATOR1,:CLOB_LOCATOR2)
```

Example: Create a CLOB_LOCATOR3 by concatenating CLOB_LOCATOR1 and CLOB_LOCATOR2.

```
"EXECSQL DECLARE LOCATOR1 VARIABLE CLOB LOCATOR"
"EXECSQL DECLARE LOCATOR2 VARIABLE CLOB LOCATOR"
"EXECSQL DECLARE LOCATOR3 VARIABLE CLOB LOCATOR"

"EXECSQL SET :locator3 = CONCAT(:locator1,:locator2)"
```

SIGNAL (Db2 V9)

RDX does not support the SIGNAL statement. Use the REXX IF-THEN-ELSE constructs or the REXX SIGNAL statements to examine the SQLCODE and pass control to the appropriate section of your REXX program.

TRUNCATE (Db2 V9)

RDX fully supports the TRUNCATE statement as illustrated in the following example.

Example:

```
"EXECSQL TRUNCATE TABLE RDX.DEMO",
 "REUSE STORAGE IGNORE DELETE TRIGGERS"
"EXECSQL COMMIT WORK"
```

The sample REXX program in the RDXSMP12 member of the CRAIEXEC library contains a working example.

UPDATE

The UPDATE statement is fully supported by RDX.

Example: Update table rows

The following example is taken from sample program RDXSMP11.

```
compnew           = 'Updated company name'
rank              = 8888
comp              = 'New company'
"EXECSQL UPDATE RDX.DEMO",
  "SET   COMPANY_NAME = :compnew,",
        "COMPANY_RANK = :rank",
  "WHERE COMPANY_NAME = :comp"
CALL ShowErrorMsg 'UPDATE'
SAY 'Updated' sqlerrd.3 'rows'
```

See the documentation and related examples for the DELETE and INSERT statements for more information.

VALUES (Db2 V9)

RDX does not support VALUES as an independent statement. However, VALUES is supported as part of a CREATE TRIGGER or SELECT statement.

VALUES INTO

RDX does not support VALUES INTO as an independent statement. However, a VALUES INTO assignment can be accomplished using a SET (or SELECT statement) as follows:

Example: Use a SET statement instead of the unsupported VALUE INTO statement.

- Not supported: `"EXECSQL VALUES(CURRENT MEMBER) INTO :MEM"`
- Use the SET statement: `"EXECSQL SET :MEM = CURRENT MEMBER"`

WHENEVER

RDX does not support the WHENEVER statement.

Db2 Instrumentation Facility Interface (IFI)

Before you request IFI services, you must first connect to a Db2 subsystem. Use the RDX ‘INIT’ and ‘TERM’ commands respectively to establish and terminate a Db2 connection. Your authorization ID must also have the privilege to issue Db2 commands or to request IFI records.

Issue Db2 Commands

You can issue Db2 commands in the ADDRESS RDX environment by using the ‘COMMAND’ prefix, as in the following example of a –DIS THREAD(*) command:

```
"COMMAND -DIS THREAD(*)"
SAY 'RC='rc 'REASON='reason

DO i              = 1 TO _cmd.0
   SAY _cmd.i
END
```

RDX externalizes the output from the Db2 command into a REXX stem named _cmd. The _cmd.0 variable contains the number of significant lines of command output within the _cmd array, as in the following example:

```
RC = 0 REASON = 00000000
DSNV401I  -DB9A DISPLAY THREAD REPORT FOLLOWS -
DSNV402I  -DB9A ACTIVE THREADS -
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
RRSAF    T        8 DB9AADMT0066 DB9AADMT ?RRSAF   0060     2
RRSAF    T    17919 DB9AADMT0001 DB9AADMT ?RRSAF   0060     3
Db2CALL  T  *     3 RAI027       RAI027   RDXPLAN  004A  4168
DISPLAY ACTIVE REPORT COMPLETE
DSN9022I  -DB9A DSNVDT '-DIS THREAD' NORMAL COMPLETION
```

Issue READS Commands

This article describes Issue READS commands.

READS commands can be issued to request that Db2 trace records be returned into the REXX stem _ifi.i. The command syntax is as follows:

```
“READS ifcid | (ifcid1, ificid2, …)”
```

One or more IFCID record types can be returned to your program on a single READS request. The stem _IFI.i contains an unpacked IFCID for each member of the data sharing group while _IFI.0 contains the number of members in the data sharing group.

To access individual IFI records and the individual fields within these records, you must do the following tasks:

- Unpack the _IFI.i variables into a set of records.
- Map the individual fields comprising these records

For information about the structure of IFI records, see the *Db2 (current supported versions) for z/OS Performance Monitoring and Tuning Guide*. IBM-supplied macros that map the IFI records reside in the *prefix*.SDSNMACS library.

Working with LOB Data

This section discusses methods of accessing and updating LOB data in Db2 tables. LOB data requires more care in handling since large objects often cannot be loaded in their entirety into memory. In addition, care is required in assembling LOB data from pieces and inserting or updating LOB data in Db2 tables. When LOBs are retrieved and broken into pieces, make sure the same algorithm is used to re-assemble it from these pieces. Otherwise, phantom characters may be introduced. This section considers several methods of accessing and changing LOB columns, as follows:

- Using LOB data types
- Using LOB locators
- Using LOB_FILE data types (V9)

Using LOB Data Types

When LOB columns are relatively small, you can simply use LOB columns as you would use VARCHAR columns. You do not need to cast a LOB column to a different data type. As supplied by the vendor, the RDX default maximum LOB size is 1 MB. However, this limit can be overridden by a DECLARE variable statement to create a REXX variable with the LOB data type that has a size *larger* than 1 MB.

RDX defines the maximum LOB size as a global default within the RDX$TSD CSECT. The supplied default is set to 1 MB, but this limit can be changed to an installation defined value by specifying a new value for the RDXMAXL# variable and reassembling the RDX$TSD CSECT. See the RDXSMP19 sample program for a working example.

Example: In this example, we define the srce variable as a CLOB of 72 bytes in length and select the column srce into this variable. The CLOB column is truncated at 80 characters (RDX adds 8 bytes for header and trailer). By default, the maximum size of the CLOB column is 1 MB.

```
"EXECSQL DECLARE srce VARIABLE CLOB(72)"
name = 'PROG001'
"EXECSQL SELECT srce INTO :srce",
 "FROM RDX.TESTLOBS",
  "WHERE NAME = :name"
```

Using LOB Locators

Because LOB values can be very large, the transfer of LOB values from the database server to host variables within client application programs can be time consuming, impractical, and sometimes impossible – owing to limitations of private region size and extremely large LOB sizes. Also, application programs typically split LOB values into pieces and process the pieces one at a time, rather than as a whole. For these cases, the application can use a large object locator (LOB locator) to reference a LOB value. A LOB locator is a host variable whose value *represents* a single LOB value in the database server. LOB locators provide a mechanism for application programs to easily manipulate very large objects without having to transfer the entire LOB value from the database server to the client application.

For example, an application can choose to handle a selected LOB value in either of the following ways:

- Select the entire LOB value and store it into an equally large host variable (as described in the previous section). This method is acceptable if the application program is going to process the entire LOB value at once.
- Select the LOB value into a LOB locator. The application program can then perform database operations on the LOB value -- *through the LOB locator*. The LOB locator value can be supplied as an input parameter to such scalar functions as SUBSTR, CONCAT, SUBSTR, and LENGTH. The LOB locator can also be used to perform assignments, search the LOB value through POSSTR, or supply the LOB locator as a parameter to a user-defined function or procedure. The output from a LOB locator operation typically is a small subset of the input LOB value.

Valid Assignments for LOB Locators

LOB locators are ordinarily used to assign data to LOB columns and retrieve data from LOB columns. However, LOB locators can also be used to assign data to *non-LOB* columns as follows:

- A CLOB or DBCLOB locator can be assigned to a CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC column. However, you cannot fetch data from CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC columns into a CLOB or DBCLOB locator.
- A BLOB locator can be assigned to a BINARY or VARBINARY column. However, you cannot fetch data from a BINARY or VARBINARY column into a BLOB locator.

Avoiding Character Conversion for LOB Locators (V9)

When assigning or fetching CLOB or DBCLOB values, it is desirable to avoid *unnecessary* character conversions between source and target fields. For this purpose, Db2 V9 introduced three new dummy tables to perform functions similar to that of SYSIBM.SYSDUMMY1. Each dummy table is associated with an encoding scheme as follows:

```
SYSIBM.SYSDUMMYA     ASCII
    SYSIBM.SYSDUMMYE    EBCDIC
    SYSIBM.SYSDUMMYU     Unicode
```

These dummy tables let you obtain the same result that can be obtained through a SET statement. For example, the following SET statement is equivalent to a SELECT INTO statement, but it avoids unnecessary character conversion:

```
SET : unicode_hv = SUBSTR(:Unicode_lob_locator,x,y)
```

The SET statement is equivalent to:

```
SELECT SUBSTR(:Unicode_lob_locator,x,y) INTO :unicode_hv
     FROM SYSIBM.SYSDUMMYU;
```

Manipulating a LOB Content without Retrieving It

Usually, you can manipulate CLOBs since they contain text. In contrast, BLOBs or DBCLOBs seldom, if ever, must be changed since they contain binary data and changes may cause unpredictable results.

Deleting Part of a LOB

The following comprehensive example illustrates how to use LOB LOCATORs to delete a substring from a LOB. Note that at no time is the content of the LOB fetched into client storage. The hypothetical CLOB being processed contains a set of 8 lines, each of which is 80 bytes in length. We wish to delete line 2 (having the content ‘PROG001 LINE 2 . . . ’) from the CLOB. Comments that are embedded within the example describe the purpose of various operations.

```
/* 1. explicitly identify data types of table columns */
/*    and host variables */
"EXECSQL DECLARE SRCE     VARIABLE CLOB LOCATOR"
"EXECSQL DECLARE LOCATOR1 VARIABLE CLOB LOCATOR"
"EXECSQL DECLARE LOCATOR2 VARIABLE CLOB LOCATOR"
"EXECSQL DECLARE LOCATOR3 VARIABLE CLOB LOCATOR"
"EXECSQL DECLARE LOCATOR4 VARIABLE CLOB LOCATOR"

/* 2.    Associate LOB value with CLOB locator */
name              = 'PROG001'
"EXECSQL SELECT SRCE INTO :LOCATOR1 FROM RDX.TESTLOBS",
 "WHERE NAME = :name"
/* 3.  Determine length of the CLOB */
```

```
"EXECSQL SET :length = LENGTH(:locator1)"
SAY 'CLOB COLUMN LENGTH='length
/* 4.  Determine starting position of LINE2 in the CLOB */
string            = 'PROG001 LINE 2'
"EXECSQL SET :beg = POSSTR(:locator1, :string)"
SAY 'BEG POSITION='beg
/* 5.  Determine starting position of LINE3 in the CLOB */
string            = 'PROG001 LINE 3'
"EXECSQL SET :end = POSSTR(:locator1, :string)"
SAY 'END POSITION='end
/* 6.  Create locator2 containing LINE1 of the CLOB */
beg1              = 1
end1              = beg - 1
"EXECSQL SET :locator2 = SUBSTR(:locator1,:beg1,:end1)"
SAY 'SUBSTR('beg1','end1')'
/* 7.  Create locator3 containing LINE3 and all other lines */
end1              = length - (end - beg)
"EXECSQL SET :locator3 = SUBSTR(:locator1,:end,:end1)"
SAY 'SUBSTR('end','end1')'
/* 8.  Create locator4 containing all lines but LINE2 */
"EXECSQL SET :locator4 = CONCAT(:locator2,:locator3)"
/* 9.  Update the CLOB that contains no LINE 2 */
"EXECSQL UPDATE RDX.TESTLOBS",
 "SET   SRCE = :LOCATOR4",
 "WHERE NAME = :name"
/* 10.  Free all locators used in this example */
"EXECSQL FREE LOCATOR (:LOCATOR1,:LOCATOR2,:LOCATOR3,:LOCATOR4)"
/* 11.  COMMIT updated CLOB */
"EXECSQL COMMIT”
```

Using LOB_FILE Data Types (V9)

LOB_FILE data types were introduced in Db2 Version 9 and were also made available to Db2 V7 and V8 through PTF. When LOB_FILE data types are referenced, Db2 copies or reads the LOB data from a sequential dataset or a PDS member. Db2 requires the RECFM of this dataset or PDS member to be V, VB, or U.

RDX makes access to LOB columns easy by internally constructing the LOB_FILE data type structure that is required by Db2. You, as a developer, must only provide the dataset name in the host variable that is associated with the LOB_FILE data type. RDX does the rest. The following examples illustrate LOB_FILE usage.

Example 1: Insert into a LOB column from a LOB_FILE dataset

First, the REXX ‘srce’ variable is declared as a CLOB FILE. Next, the name of the partitioned dataset and member that contains the data to be inserted into the LOB column is assigned to the ‘srce’ variable. Finally, the value of the LOB1 member from the TEST.LOB.IN dataset is INSERT’d into the SRCE column of the table RDX.TESTLOBS.

```
"EXECSQL DECLARE srce VARIABLE CLOB FILE"
srce = "TEST.LOB.IN(LOB1)"
"EXECSQL INSERT INTO RDX.TESTLOBS",
 "(SRCE)",
  "VALUES(:srce)"
```

Example 2: Select a LOB column into a LOB_FILE dataset

As in Example 1, the ‘srce’ variable is declared as a CLOB FILE. The name of the target PDS and member is assigned to the host variable ‘srce’ If the dataset does not exist, Db2 allocates it on a work DASD volume. The contents of the ‘srce’ column are copied into the specified PDS member using a SELECT INTO statement as follows:

```
"EXECSQL DECLARE srce VARIABLE CLOB FILE"
srce = "TEST.LOB.OUT(LOB1)"
"EXECSQL SELECT srce INTO :srce",
 "FROM RDX.TESTLOBS",
  "WHERE NAME = :name",
   "FETCH FIRST 1 ROWS ONLY"
```

The sample REXX program in the RDXSMP17 member of the CRAIEXEC library contains a working example.

Working with pureXML (Db2 V9)

pureXML was introduced in Db2 9 for z/OS to enable applications to manage XML data residing in Db2 tables. You can store well-formed XML documents in their hierarchical form based on the XQuery and XPath data model (XDM) and retrieve all or portions of those documents.

RDX simplifies access to XML columns in Db2 tables by implicitly casting XML data types as either LOB or character data types. On one hand, you INSERT, UPDATE, and DELETE rows of Db2 tables containing XML data as you would any other data type. On the other hand, you can use XQuery expressions with XML scalar functions provided by Db2 to parse the contents of XML column data and combine the results of the parsing with SQL predicates.

Review the various XML processing scenarios and examples. This information considers only issues that are unique to RDX.

The IBM publication *Db2 11 for z/OS pureXML Guide - SC19-4064* describes pureXML more thoroughly and provides samples that are written in C, C++, and Java. This documentation extends the pureXML Guide with samples that are written in the REXX language (which is presently not supported by IBM) and documents the RDX implementation of the REXX interface to pureXML on z/OS.

The sample REXX programs RDXSMP21 and RDXSMP22 member of the CRAIEXEC library contains working examples.

Specifying XQuery expressions

XQuery expressions may contain single (‘) and double (“) quote characters as well as open ([ ) and close ( ]) bracket characters. An XML document can be formatted with carriage return, new line, and tabs. Make sure to use the correct codes for these special characters when translating between ASCII, EBCDIC, and UNICODE CCSIDs.

When creating XQuery expressions, it may be convenient to see the special characters which must be altered in order for the XQuery expression to be correct. The following table shows the hexadecimal codes for these special characters in their ‘displayable’ and ‘executable’ forms:

Special Character	Displayable hex code	Executable hex code	Alternative specification
[- opening bracket	‘AD’x	‘BA’x	??(
] - closing bracket	‘BD’x	‘BB’x	??)
¬ - circumflex	‘5F’x	‘B0’x	

Example: XQuery expression embedded within SQL statement.

```
"EXECSQL SELECT info INTO :info FROM MYCUSTOMER",
 "WHERE XMLEXISTS('declare default element namespace",
                  '"http://posample.org";',
                  "/customerinfo[@cid = $c]'",
                  'passing INFO, CAST(:cid AS INTEGER) AS "c")'
```

The opening ([ ) and closing ( ]) brackets appearing in the expression in the example are not valid in an EBCDIC string. They cannot be specified as shown, unless your keyboard supports the correct codes that are associated with these special characters (as shown in the ‘Executable hex code’ column of the table). For your convenience, RDX provides an EDIT macro that is named RDXBRACK to translate these special characters between their displayable and executable forms. The syntax of the RDXBRACK edit macro is:

```
RDXBRACK mode
```

If the mode is set to 'SHOW', then special symbols are translated from their ‘executable’ code to a ‘displayable’ form. If the mode is not ‘SHOW’, then reverse translation takes place.

Care is required to code XQuery statements and represent them correctly. We recommend that you use REXX single (‘) and double (“) quotes to combine substrings of an XQuery expression into a resultant string in such a way that XQuery’s own single and double quotes are segregated into independent substrings. The example illustrates how to do so properly. Admittedly, this approach is tricky and requires some practice. However, it permits you to specify any possible XQuery expression within a REXX program.

Tutorial of pureXML

The DDL and SQL queries in the RDXXML member of the RDXCNTL library provide first-hand experience with XML. The SPUFI that is supplied with Db2 9 for z/OS can create a table with XML columns but cannot *display* XML column data. Alternatively, you can use another product - Db2 eXtensions/SPUFI to both execute the DDL and SQL statements within RDXXML and to display the XML column results. A trial copy of DX/SPUFI is supplied along with RDX.

Using Host Variables in XML Scalar Functions

Consider the following REXX code fragment:

```
xmldoc =,         /* serialized XML document */
'<customerinfo xml:space="default" xmlns="http://posample.org"',
"Cid='1009'> <name>Kathy Smith</name> <addr country='Canada'>",
"<street>15 Rosewood</street> <city>Toronto</city>",
"<prov-state>Ontario</prov-state> <pcode-zip>M6W 1E6</pcode-zip>",
"</addr> <phone type='work'>416-555-4444</phone> </customerinfo>"
```

The XML document that is stored in the REXX host variable xmldoc is being inserted into a table with an XML column in the following examples. Example 1 illustrates how *not* to use host variables with XML scalar functions while Example 2 shows a coding technique involving casting that executes without errors:

Example 1: Improper way to use REXX host variables in XML scalar functions.

```
"EXECSQL INSERT INTO MYCUSTOMER (Cid, Info)",
 "VALUES(1009,",
 "XMLPARSE(DOCUMENT :xmldoc STRIP WHITESPACE))"
```

The INSERT statement fails with the following error message:

```
DSNT408I SQLCODE = -418, ERROR:  A STATEMENT STRING TO BE PREPARED CONTAINS
 AN INVALID USE OF PARAMETER MARKERS
DSNT418I SQLSTATE   = 42610 SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNXOBFF SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD    = 30  0  0  -1  0  0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD    = X'0000001E'  X'00000000'  X'00000000'  X'FFFFFFFF'
         X'00000000'  X'00000000' SQL DIAGNOSTIC INFORMATION
TERM         - SQLCODE = 0 RC = 0 REASON = 00000000
```

Example 2: The correct way to use REXX host variables in XML scalar functions.

The INSERT statement in Example 1 is modified to include the CAST scalar function. This modification permits the INSERT to execute without errors.

```
"EXECSQL INSERT INTO MYCUSTOMER (Cid, Info)",
 "VALUES(1009,",
 "XMLPARSE(DOCUMENT CAST(:xmldoc AS CLOB(1M)) STRIP WHITESPACE))"
```

Example 3: You can embed host variables within an XQuery statement as shown:

```
cid               = 1010
"EXECSQL UPDATE MYCUSTOMER",
 "SET Info =:xmldoc",
  "WHERE XMLEXISTS('declare default element namespace",
                   '"http://posample.org";',
                   "/customerinfo[@cid = $c]'",
                   'passing INFO, CAST(:cid AS INTEGER) AS "c")'
```

In the example, the host variable ‘cid’ assigns a value to the ‘cid’ attribute variable of the /customerinfo document.

Example of Functions to Construct XML Values

The following SQL/XML functions can be used separately or together to publish relational data in XML format.

- **XMLAGG aggregate function**
  XMLAGG returns an XML sequence containing an item for each non-null value in a set of XML values
- **XMLATTRIBUTES scalar function**
  XMLATTRIBUTES constructs XML attributes from its arguments. XMLATTRIBUTES can only be used as an argument of the XMLELEMENT function.
- **XMLCOMMENT scalar function**
  XMLCOMMENT returns an XML value with a single comment node, with the input argument as the content.
- **XMLCONCAT scalar function**
  XMLCONCAT returns a sequence containing the concatenation of a variable number of XML input arguments.
- **XMLDOCUMENT scalar function**
  XMLDOCUMENT returns an XML value with a single document node containing zero or more child nodes. XMLDOCUMENT creates a document node (which by definition, every XML document must have). A document node is not visible in the serialized representation of XML. However, every document that is to be stored in a Db2 table must contain a document node. Note that the XMLELEMENT function does not create a document node, only an element node. Thus, when constructing XML documents that are to be inserted, it is not sufficient to create just an element node. The document must also contain a document node.
- **XMLELEMENT scalar function**
  XMLELEMENT returns an XML value that is an XML element node.
- **XMLFOREST scalar function**
  XMLFOREST returns an XML value that is a sequence of XML element nodes.
- **XMLNAMESPACES declaration**
  The XMLNAMESPACES function constructs namespace declarations from its arguments. This declaration can only be used as an argument to either the XMLELEMENT or XMLFOREST function.
- **XMLPI scalar function**
  XMLPI returns an XML value with a single processing instruction node.
- **XMLTEXT scalar function**
  XMLTEXT returns an XML value with a single text node having the input argument as the content.
  You can combine these scalar functions to construct XML values that contain different types of nodes. Specify the functions in the order in which the corresponding elements should appear.

Example:

Suppose that you want to construct the following document, which has constant values:

```
<elem1 xmlns="http://posample.org" id="111">
  <!-- example document -->
  <child1>abc</child1>
```

```
  <child2>def</child2>
</elem1>
```

The following SELECT statement constructs the required document in the host variable ‘XMLDOC’:

```
'EXECSQL SELECT XMLELEMENT (NAME "elem1",',
  "XMLNAMESPACES (DEFAULT 'http://posample.org'),",
  "XMLATTRIBUTES ('111' AS" '"id"),',
  "XMLCOMMENT (' example document '),",
  "XMLFOREST('abc' as" '"child1",',
  "'def' as" '"child2"))',
  "INTO :xmldoc",
  "FROM SYSIBM.SYSDUMMY1"
```

Note the use of single and double quotes to combine parts of the SQL statement string into a valid statement, that in turn contains both single and double quoted strings.

Example of XML Scalar Functions and Predicates

This section reviews all XML scalar functions that were implemented in Db2 V9.1 for z/OS. REXX code examples are presented along with brief explanations of the code.

- **XMLEXISTS predicate**
  The XMLEXISTS predicate can be used to restrict the set of rows that are returned by a query based on the evaluation of XML column values by an embedded XQuery expression. If the XQuery expression returns an empty sequence, the value of the XMLEXISTS predicate is false. Otherwise, XMLEXISTS returns true. Rows that correspond to an XMLEXISTS value of true are returned. For example, consider the following UPDATE statement:
  Example:
  ```
  cid = 101
  "EXECSQL UPDATE MYCUSTOMER",
   "SET Info =:xmldoc",
    "WHERE XMLEXISTS('declare default element namespace",
                     '"http://posample.org";',
                     "/customerinfo[@cid = $c]'",
                     'passing INFO, CAST(:cid AS INTEGER) AS "c")'
  ```
  In this example, the XQuery expression selects a row in which the INFO column’s XML document has the attribute: cid =”101”. This row’s INFO column is then updated with the value assigned to the ‘xmldoc’ host variable.
- **XMLPARSE function**
  The XMLPARSE function parses the argument as an XML document and returns an XML value that adheres to the XQuery data model. XMLPARSE allows you to strip or preserve white spaces (new lines, carriage return, tabs, blanks, and so on). These characters can be preserved to allow serialized XML document formatting, before a visual examination.
  Example: Insert the XML document that is stored in the REXX host variable ‘xmldoc’ and preserve white spaces in the XML value.
  ```
  "EXECSQL INSERT INTO MYCUSTOMER (cid, Info)",
    "VALUES(:cid,",
    "XMLPARSE(DOCUMENT CAST(:xmldoc AS CLOB(1M)) PRESERVE WHITESPACE))"
  ```
  After the XML document is inserted, it can be retrieved (or serialized) and formatted in accordance with the new line, carriage return, and tab characters that are saved in the XML document.
- **XMLQUERY function**
  XMLQUERY lets you execute an XQuery expression from within an SQL context. You can pass variables to the XQuery expression specified within the XMLQUERY. XMLQUERY returns an XML value, which is an XML sequence

that can be empty or can contain one or more items. When you execute XQuery expressions from within an XMLQUERY function, you can:

– Retrieve parts of stored XML documents, instead of entire XML documents.
– Enable XML data to participate in SQL queries.
– Operate on both relational and XML data in the same SQL statement.
– Apply further SQL processing to the returned XML values (for example, ordering results with the ORDER BY clause of a subselect), after you use XMLCAST to cast the results to a non-XML type.

XQuery is case-sensitive, so you must ensure that the case of variables that are specified in an XMLQUERY function and in its XQuery expression match exactly.

For example, using the MYCUSTOMER table, execute the following XMLQUERY:

```
"EXECSQL SELECT CID, XMLQUERY('declare default element namespace",
  "http://posample.org"; /customerinfo/phone' passing INFO)",
  'AS "PHONE FROM INFO"',
  "FROM CUSTOMER WHERE CID IN (102,104)"
```

- **XMLSERIALIZE function**

  The XMLSERIALIZE function directs the Db2 database server to perform XML serialization *before* it sends XML data to the client application. This process is called *explicit serialization*. Alternatively, you can omit the XMLSERIALIZE call, and retrieve data from an XML column directly into application variables. The Db2 database server then serializes the XML data during retrieval, a process called *implicit serialization*.

  Implicit serialization is usually the preferred method. Sending XML data to the client allows the Db2 client to handle the XML data properly. Explicit serialization requires additional handling by the client.

  Example:

  The XML document that is stored in the 'info' column is serialized into a string that is placed into the CLOB host variable that is named 'info'. The serialized string has ASCII encoding. The REXX function A2E (supplied with RDX) translates this ASCII string to EBCDIC so the value can be displayed in a TSO/ISPF session.

```
"EXECSQL SELECT XMLSERIALIZE(info as BLOB(1M))",
 "INTO :info FROM MYCUSTOMER WHERE cid = :cid"
CALL ShowErrorMsg 'SELECT'
IF sqlcode       \= 0 THEN RETURN sqlcode
info = A2E(info)
CALL DisplayXML
```

- **DSN_XMLVALIDATE function**

  The SYSFUN. DSN_XMLVALIDATE function is used to validate an XML document as follows:

  – Determine whether the structure, content, and data types of an XML document are valid.
  – Strip ignorable whitespace from the input document.

  Before using the DSN_XMLVALIDATE function, all the schema documents that make up the XML schema to be validated must first be registered in the built-in XML schema registry (XSR).

  DSN_XMLVALIDATE can only be invoked as an argument of the XMLPARSE function. When you invoke DSN_XMLVALIDATE within XMLPARSE, you must specify the STRIP WHITESPACE option for XMLPARSE.

```
"EXECSQL INSERT INTO MyProduct",
 "(pid, name, Price, PromoPrice, PromoStart, PromoEnd,",
 "description)",
 "VALUES( '110-100-01','Anvil', 9.99, 7.99,",
 "'11-02-2004','12-02-2004',",
 "XMLPARSE(DOCUMENT SYSFUN.DSN_XMLVALIDATE(CAST ? AS CLOB),",
 "'SYSXSR.PRODUCT')))"
```

SQL Communication Area (SQLCA)

The SQLCA is a structure or collection of variables that is updated after each SQL statement executes. RDX creates an SQLCODE variable after every SQL statement executes to indicate the success or failure of the statement. By default, RDX externalizes the entire SQLCA structure. Doing so can be wasteful, especially when the SQLCODE is zero.

You can use the CNTL DSNTIAR NO command to suppress the automatic externalization of the SQLCA into REXX variables. Instead, you can explicitly issue a CNTL DSNTIAR CALL command to examine the SQLCA, even if the SQLCODE is zero. When the SQLCODE is not zero, RDX automatically externalizes the SQLCA.

The following list illustrates the SQLCA variables following an error in SQL statement processing:

```
SQLCODE..................... -204
SQLERRP..................... DSNXOTL
SQLERRD.1................... -500
SQLERRD.3................... 0
SQLERRD.2................... 0
SQLERRD.6................... 0
SQLERRD.5................... 0
SQLERRD.4................... -1
MENU0002.................... Test Dialogs

MENU0001.................... TEST
SQLSTATE.................... 42704
SQLWARN.0...................
SQLWARN.2...................
SQLWARN.1...................
SQLWARN.4...................
SQLWARN.6...................
SQLWARN.5...................
SQLWARN.8...................
SQLWARN.10..................
SQLWARN.9...................
SQLWARN.7...................
SQLWARN.3...................
SQLERRMC.................... SYSIBM.IPLIST2
```

The IBM “SQL Reference” describes the variables that comprise the SQLCA.

SQL Descriptor Area (SQLDA)

The SQLDA is a data structure that is required for execution of the following SQL statements: DESCRIBE, PREPARE INTO, OPEN USING DESCRIPTOR, FETCH, EXECUTE, and CALL.

The meaning of the information in the SQLDA depends on the context in which it is used. For DESCRIBE and PREPARE INTO statements, Db2 sets the fields in the SQLDA to provide information to the application program. In contrast, the OPEN USING DESCRIPTOR, FETCH, EXECUTE, and CALL statements require the application program to supply the SQLDA information to Db2. The SQLDA starts with a header that defines the type of SQLDA (SQLDAID) and specifies the number of variable descriptors (SQLVAR). Each SQLVAR occurrence describes either a column of a query result, an input parameter marker, or a result set locator--depending on the type of SQLDA.

Db2-Returned SQLDA

Db2 fills the SQL Descriptor Area (SQLDA) with information as part of processing the SQL DESCRIBE and PREPARE INTO statements. RDX maps the SQLDA that is populated by Db2 into a set of REXX variables that share the user-designated stem that is specified through the INTO :stem clause of the DESCRIBE or PREPARE statement. The last node of a compound variable's name (such as stem.i.sqltype) corresponds to the variable names that are documented in the *SQL Reference* for the DSNREXX interface. The sample program in the RDXSQLDA member of the CRAIEXEC library writes out SQLDA variables depending on the SQL statement that created the SQLDA.

The following examples illustrate the SQLDA output that is produced by various types of DESCRIBE statements.

Example 1: PREPARE and DESCRIBE OUTPUT

This example was taken from sample program RDXSMP10.

```
stm = 'SELECT * FROM SYSIBM.IPNAME WHERE LINKNAME = :linkname'
"EXECSQL PREPARE S2 FROM :stm"
"EXECSQL DESCRIBE S2 INTO :cat"
CALL GLOBVAR 'PUT','cat.','STEM'
      CALL RDXSQLDA 'DESCRIBE','cat'
```

The resulting SQLDA displays as follows:

```
CAT.SQLDAID................... = SQLDA
CAT.SQLD...................... = 3
---- Column 1 -----------------------------------------
CAT.1.SQLTYPE................. = 448
CAT.1.SQLLEN.................. = 24
CAT.1.SQLCCSID................ = 1208
CAT.1.SQLNAME................. = LINKNAME
CAT.1.SQLLABEL................ = LINKNAME
CAT.1.SQLDTSN................. = VC
CAT.1.SQLDTLN................. = VARCHAR
---- Column 2 -----------------------------------------
CAT.2.SQLTYPE................. = 448
CAT.2.SQLLEN.................. = 254
CAT.2.SQLCCSID................ = 1208
CAT.2.SQLNAME................. = IPADDR
CAT.2.SQLLABEL................ = IPADDR
CAT.2.SQLDTSN................. = VC
CAT.2.SQLDTLN................. = VARCHAR
---- Column 3 -----------------------------------------
```

```
CAT.3.SQLTYPE................. = 452
CAT.3.SQLLEN.................. = 1
CAT.3.SQLCCSID................ = 1208
CAT.3.SQLNAME................. = IBMREQD
CAT.3.SQLLABEL................ = IBMREQD
CAT.3.SQLDTSN................. = C
CAT.3.SQLDTLN................. = CHAR
```

NOTE
RDX provides the REXX exec that is named RDXSQLDA that can display all possible SQLDA types. In the example above, the GLOBVAR 'PUT' function is used to pass REXX variables to the external subroutine RDXSQLDA. The RDXSQLDA exec can then retrieve these variables using a GLOBVAR 'GET' function call.

Example 2: DESCRIBE INPUT

This example also comes from the RDXSMP10 sample program. Assume that the statement S2 was prepared as in the Example 1.

```
"EXECSQL DESCRIBE INPUT S2 INTO :in"
CALL GLOBVAR 'PUT','in.','STEM'
CALL RDXSQLDA 'DESCRIBE','in'
```

The resulting SQLDA displays as follows:

```
IN.SQLDAID.................... = SQLDA
IN.SQLD....................... = 1
---- Column 1 -----------------------------------------
IN.1.SQLTYPE.................. = 449
IN.1.SQLLEN................... = 24
IN.1.SQLCCSID................. = 1208
IN.1.SQLNAME.................. = RDXV001
IN.1.SQLLABEL................. = RDBV001
IN.1.SQLDTSN.................. = VC
IN.1.SQLDTLN.................. = VARCHAR
```

Example 3: DESCRIBE PROCEDURE

This example comes from the sample program RDBSMP05. The DESCRIBE PROCEDURE request should only be issued after a CALL procedure statement executes successfully. In this example, the called stored procedure returns one result set:

```
"EXECSQL DESCRIBE PROCEDURE :procname INTO :out"
CALL GLOBVAR 'PUT','out.','STEM'
CALL RDXSQLDA 'PROCEDURE','out'
```

The resulting SQLDA displays as follows:

```
OUT.SQLDAID................... = SQLPR
OUT.SQLD...................... = 1
---- Column 1 -----------------------------------------
OUT.1.SQLDATA................. = 00000001
OUT.1.SQLLOCATOR.............. = 00000001
OUT.1.SQLNAME................. = C1
```

Example 4: DESCRIBE CURSOR

This example comes from the RDXSMP05 sample program.

```
"EXECSQL DESCRIBE CURSOR C101 INTO :csr"
```

```
CALL GLOBVAR 'PUT','csr.','STEM'
CALL RDXSQLDA 'DESCRIBE','csr'
```

The resulting SQLDA displays as follows:

```
CSR.SQLDAID................... = SQLRS
CSR.SQLD...................... = 13
---- Column 1 -----------------------------------------
CSR.1.SQLTYPE................. = 452
CSR.1.SQLLEN.................. = 20
CSR.1.SQLCCSID................ = 1140
CSR.1.SQLNAME................. = COMPANY_NAME
CSR.1.SQLLABEL................ = COMPANY_NAME
CSR.1.SQLDTSN................. = C
CSR.1.SQLDTLN................. = CHAR
---- Column 2 -----------------------------------------
CSR.2.SQLTYPE................. = 500
CSR.2.SQLLEN.................. = 2
CSR.2.SQLCCSID................ = 0
CSR.2.SQLNAME................. = COMPANY_RANK
CSR.2.SQLLABEL................ = COMPANY_RANK
CSR.2.SQLDTSN................. = IS
CSR.2.SQLDTLN................. = SMALLINT
---- Column 3 -----------------------------------------
CSR.3.SQLTYPE................. = 452
CSR.3.SQLLEN.................. = 20
CSR.3.SQLCCSID................ = 1140
CSR.3.SQLNAME................. = CEO
CSR.3.SQLLABEL................ = CEO
CSR.3.SQLDTSN................. = C
CSR.3.SQLDTLN................. = CHAR
---- Column 4 -----------------------------------------
CSR.4.SQLTYPE................. = 384
CSR.4.SQLLEN.................. = 10
CSR.4.SQLCCSID................ = 1140
CSR.4.SQLNAME................. = FOUNDED_DATE
CSR.4.SQLLABEL................ = FOUNDED_DATE
CSR.4.SQLDTSN................. = D
CSR.4.SQLDTLN................. = DATE
```

User-Created SQLDA

You might want to define the SQLDA manually that is referenced in the USING DESCRIPTOR clause of the SQL CALL statement. This manual definition might be necessary because the SQLDA that is produced by a DESCRIBE PROCEDURE statement describes the result sets that are returned by the stored procedure. However, that SQLDA does *not* describe the procedure's input and output parameters.

As such, you should manually assign values to REXX stem variables to construct a SQLDA that describes the input and output parameters of the stored procedure. When the SQL CALL statement which references this SQLDA is executed, RDX fetches the REXX stem variables that you specify to construct an internal SQLDA that describes the called procedure's parameters. For example, consider the following code fragment:

```
/* initialize SQLDA stem sp. */
```

```
sp.SQLDAID        = 'SQLDA   '
sp.SQLD           = 3

sp.1.SQLTYPE      = 448
sp.1.SQLLEN       = 254
sp.1.SQLNAME      = 'table'
sp.1.SQLDATA      = 'RDX.DEMO'

sp.2.SQLTYPE      = 496
sp.2.SQLLEN       = 4
sp.2.SQLNAME      = 'rank'
sp.2.SQLDATA      = 5

sp.3.SQLTYPE      = 448
sp.3.SQLLEN       = 254
sp.3.SQLNAME      = 'message'
sp.3.SQLDATA      = ''

/* assign parameter values */

procname          = 'RDX.RDBTSP1'
table             = 'RDX.DEMO'
rank              = 3

/* invoke the stored procedure */

"EXECSQL CALL :procname USING DESCRIPTOR :sp"
```

The REXX stem sp. is initialized with the values of required SQLDA variables. The SQLDA contains three SQLVAR entries. SQLNAME variables are initialized with values to be used by the CALL statement. The RDXSMP08 member of the CRAIEXEC library contains a working example of calling a stored procedure.

Messages and Codes

This section contains information about messages and return and reason codes.

After a RDX statement executes, the following variables are always set:

RC - The return code variable RC describes the result of command execution as follows:

- **0** - Operation successful. After a RDX INIT call, the Db2VRM and Db2SSN variables are set to the Db2 version (for example, 810) and Db2 subsystem name (for example, DSN8), respectively.
- **1** - SQLCODE > 0 was received. Review any SQL warning messages that were returned from the SQLERR.i stem. For the RDX TERM command, RC=1 means there was no active connection to Db2 when the TERM command was issued.
- **-1** - SQLCODE < 0 was received. Review any SQL error messages that were returned from the SQLERR.i stem.
- **-3** - Invalid first command token. The first token must be one of the following RDX supported commands: INIT, TERM, EXECSQL, COMMAND, READS, READA, or WRITE.
- **-4** - Failure occurred during RDX initialization.
- **-100nnn** - RDX issued an error message whose numeric suffix is 'nnn'. Review the text of the RDX error message that were returned from the SQLERR.i stem.
- **04, 08, 12** - These return codes are set after a RDX INIT call fails to connect to the target Db2 subsystem. Examine the REASON variable and check the reason code documentation that appears in the Db2 publication *Db2 for z/OS Codes*.
- **REASON** - The reason code variable REASON qualifies the value of the return code variable RC. For a RDX INIT failure, the reason code is set by the Db2 attachment interface to indicate the reason for the connection failure.
- **1** - This reason code is set when the first command token is invalid, as described by RC = -3.
- **SQLCODE** - contains the SQL return code that is associated with executing the latest SQL statement. The SQLCODE contains a RDX message ID when the RDX parser detects a syntactic or semantic error.

Messages

By default, all RDX messages are available in memory and can be displayed by the programmer when required. After statement execution, if the SQLCODE is not zero, usually you see a message describing the error. RDX places the message in the REXX stem named SQLERR.i, which can be displayed with the following statements:

```
DO i = 1 TO SQLERR.0
   SAY SQLERR.i
END
```

When the SQLCODE is zero, you may want to translate the SQLCA into a meaningful Db2 message. To do so, issue the following command:

```
CNTL DSNTIAR CALL
```

In response, RDX populates the SQLERR.i stem with the SQL error message that is formatted by the IBM-supplied DSNTIAR program. Electing to call the DSNTIAR routine explicitly and only when needed improves performance because doing so avoids the unnecessary creation of SQLERR.i stem variables when the contents of the SQLCA is of no interest.

RDX001I REXX to Db2 interface Version *version*

Explanation: Informational message indicating the RDX version.

Response: None

RDX002E Token *token* is not recognized

Explanation: The RDX parser encountered a token *token* that should not occur in the statement that is being parsed.

Response: Review the SQL statement preceding this message. Verify the statement syntax and correct the *token*

RDX003E USQ did not initialize – severe error

Explanation: The Universal SQL Attachment component that is used by RDX failed to initialize.

Response: This error is a program logic error and should be reported to Broadcom Support.

RDX004E You passed an empty string as a RDX command

Explanation: You specified an empty string as a host command while ADDRESS RDX was active.

Response: If you are unsure which host command was the empty string, use the REXX TRACE ‘C’ command to display all host commands as they execute. Locate the last host command executed before this message is issued and correct it.

RDX006E EXECSQL verb *verb* action routine was not defined

Explanation: The first token in the RDX command string is not a valid SQL statement that is supported by RDX.

Response: If the *verb* is a valid SQL statement, report the problem to Broadcom Support.

RDX007E SET :hv = statement requires a host variable that was not specified

Explanation: The syntax of the SET statement is incorrect.

Response: Correct the syntax of the SET statement and re-execute the program.

RDX008E After “=” only USER, CURRENT, SUBSTR, CONCAT, POSSTR or LENGTH keywords are allowed

Explanation: You specified a SET statement where after the ‘=’ an invalid keyword was encountered.

Response: Correct the syntax of the SET statement and re-execute the program.

RDX009E Invalid (unsupported) CURRENT register

Explanation: You specified an unsupported CURRENT register value.

Response: Correct the CURRENT REGISTER specification and re-execute the program. If the CURRENT register specification is correct, report the problem to Broadcom Support.

RDX010E Syntax SET CONNECTION :hv

Explanation: Your SET CONNECTION statement does not conform to the statement syntax appearing in the message text.

Response: Correct the syntax of the SET statement and re-execute the program.

RDX011E Syntax: SET CONNECTION . . . | CURRENT . . .

Explanation: You violated the syntax of the SET statement.

Response: Correct the syntax of the SET statement and re-execute the program.

RDX012E Syntax: SET current_register = :hv

Explanation: You violated the syntax of the SET statement.

Response: Correct the syntax of the SET statement and re-execute the program.

RDX013E Syntax: SET CURRENT OPTIMIZATION HINT = :HV

Explanation: You violated the syntax of the SET statement.

Response: Correct the syntax of the SET statement and re-execute the program.

RDX014E Syntax: SET CURRENT LOCALE LC_CTYPE = :HV

Explanation: You violated the syntax of the SET statement.

Response: Correct the syntax of the SET statement and re-execute the program.

RDX015E Invalid syntax of the SQL statement

Explanation: You violated the syntax of the SQL statement.

Response: Correct the syntax of the SQL statement and re-execute the program.

RDX016E Invalid value of LOCATOR variable = *locator*

Explanation: You violated the syntax of the SQL statement.

Response: Correct the syntax of the SQL statement and re-execute the program.

RDX017E Syntax: EXECUTE Sn {USING DESCTIPTOR :stem}, n=1-100 -- but detected *value*

Explanation: You violated the syntax of the EXECUTE statement.

Response: Correct the syntax of the EXECUTE statement and re-execute the program.

RDX018E Syntax: DESCRIBE Sn INTO :SQLDA {USING BOTH|ANY|LABELS} – but detected *value*

Explanation: You violated the syntax of the SQL DESCRIBE statement – the token *token* is not correct.

Response: Correct the syntax of the DESCRIBE statement and re-execute the program.

RDX019E Syntax: DESCRIBE TABLE :name INTO :SQLDA

Explanation: You violated the syntax of the SQL DESCRIBE TABLE statement.

Response: Correct the syntax of the DESCRIBE TABLE statement and re-execute the program.

RDX020E Syntax: DESCRIBE PROCEDURE :name INTO :SQLDA

Explanation: You violated the syntax of the DESCRIBE PROCEDURE statement.

Response: Correct the syntax of the DESCRIBE PROCEDURE statement and re-execute the program.

RDX021E Syntax: DESCRIBE CURSOR :name INTO :SQLDA

Explanation: You violated the syntax of the DESCRIBE CURSOR statement.

Response: Correct the syntax of the DESCRIBE CURSOR statement and re-execute the program.

RDX022E Statement requires no SQLDA – EXECUTE IMMEDIATE

Explanation: You specified an SQLDA on the EXECUTE statement but the prepared statement has no host variables.

Response: Use EXECUTE IMMEDIATE rather than EXECUTE USING SQLDA for the current SQL statement.

RDX023E Syntax: PREPARE Sn INTO :SQLDA {USING BOTH|ANY|LABELS| ATTRIBUTES :ATTR:IV FROM :STMT -- *token*

Explanation: Your PREPARE statement does not conform to the statement syntax appearing in the message text. *token* is not correct.

Response: Correct the PREPARE statement and re-execute.

RDX024E Syntax: DECLARE name VARIABLE DATATYPE(sqltype) LENGTH(length) | LENGTH(p,s) {INPUT} {INDICATOR} -- *token*

Explanation: The DECLARE statement violates the syntax appearing in the message text. *token* is not correct.

Response: Correct the DECLARE statement and re-execute.

RDX025E Syntax: DESCRIBE INPUT Sn INTO :SQLDA -- *token*

Explanation: The DESCRIBE statement violates the syntax appearing in the message text. *token* is not correct.

Response: Correct the DESCRIBE statement and re-execute.

RDX026E Syntax: CALL :proc (:hv1, hv2, . . .) -- *token*

Explanation: The CALL statement violates the syntax appearing in the message text. *token* is not correct.

Response: Correct the CALL statement and re-execute.

RDX027E Fetch variable *varname* failed RC=*retcode* – *operation*

Explanation: The fetch of the REXX variable named *varname* (which is used as a host variable in the current SQL statement) has failed for the operation *operation.*

Response: If *varname* is used as an input host variable, make sure it was previously defined in your REXX exec and was assigned a value with the correct data type.

RDX028E Cursor name: Cn, n=1,200, statement name: Sn, n=1,100 -- *token*

Explanation: The current SQL statement uses a cursor or statement name which is syntactically incorrect. The incorrect value is shown as *token*.

Response: Correct the syntax error and re-execute your program.

RDX029E Syntax: EXECUTE IMMEDIATE :sqlstmt -- *token*

Explanation: The EXECUTE IMMEDIATE statement violates the syntax appearing in the message text. *token* is not correct.

Response: Correct the SQL statement syntax and re-execute the program.

RDX030E Statement Sn was not prepared – PREPARE then EXECUTE

Explanation: You attempted to execute statement Sn that was not previously prepared. Alternatively, an explicit or implicit COMMIT or ROLLBACK was issued that invalidated the prepared SQL statement.

Response: Review the program logic to make sure you issued PREPARE Sn and there was no intervening COMMIT or ROLLBACK statement issued. Also, make sure no negative SQLCODE was received, such as -911, that caused Db2 to issue an implicit ROLLBACK.

RDX031E SQL statement contains input parameter markers (?) but DESCRIBE INPUT was not issued

Explanation: This message is self-explanatory.

Response: Issue DESCRIBE INPUT for this statement before EXECUTE.

RDX032E Output SQLDA is absent – you either did not DESCRIBE it or the SQL statement is not a SELECT

Explanation: You issued a SELECT statement while the output SQLDA has no entries.

Response: This condition can occur when a non-SELECT SQL statement is prepared (such as an INSERT, UPDATE, or DELETE) and an OPEN request is issued for the statement. Review the program logic to make sure an OPEN is issued for a SELECT statement.

RDX033E Number of SQLD vars *var#* not the same as the number of output host variables *hostvar#* found in SQL stmt

Explanation: The number of used entries in the SQLDA does not match the number of host variables in the prepared SQL statement

Response: This error may occur when the wrong SQLDA is specified for the currently executing SQL statement. A DESCRIBE INTO request may have been issued for a different SQL statement.

RDX034E Syntax: OPEN Cn {USING DESCRIPTOR :SQLDA},n=1-100

Explanation: The OPEN statement violates the syntax appearing in the message text.

Response: Correct the syntax of the SQL OPEN statement and re-execute the program.

RDX035E Syntax: CLOSE Cn, n=1-100

Explanation: The CLOSE statement violates the syntax appearing in the message text. *token* is not correct.

Response: Correct the syntax of the SQL CLOSE statement and re-execute the program.

RDX036E Syntax: FETCH FROM Cn {options}, n=1-200 -- *token*

Explanation: The FETCH statement violates the syntax appearing in the message text. *token* is not correct.

Response: Correct the syntax of the SQL FETCH statement and re-execute the program.

RDX037E SQL statement requires more host variables than were actually specified

Explanation: The SQLDA of the prepared SQL statement contains fewer SQLD entries than the number of host variables specified in the SQL statement.

Response: This message is probably due to a mismatched SQLDA and the Sn or Cn with which it is used. Correct the error and re-execute your program.

RDX038E There are more host variables in SQL statement than were found in prepared SQLDA – *hostvar#*

Explanation: The prepared SQLDA (explicitly specified or implicitly created by RDX) does not match the number of host variables in the SQL statement.

Response: This message is probably due to a mismatched SQLDA and the Sn or Cn with which it is used. Correct the error and re-execute your program.

RDX039E Token Scan Table (TTS) is too small for the SQL statement -- Sn

Explanation: This message indicates a program logic error.

Response: Report this problem to Broadcom Support.

RDX040E Statement / Cursor *name* must be within the range of *start* and *end*

Explanation: The statement or cursor name has the form Sn/Cn where n is limited by the specified *start* and *end* range.

Response: Rename the statement or cursor to be within the requested range.

RDX041E Syntax: SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION

Explanation: The SET statement violates the syntax appearing in the message text.

Response: Correct the syntax of the SQL SET statement and re-execute the program.

RDX042E DECLARE statement is incorrect -- *token*

Explanation: The syntax of the DECLARE statement was violated at the *token*.

Response: Correct the syntax error and re-execute your program.

RDX043E DECLARE statement has no SELECT clause

Explanation: The DECLARE statement requires a SELECT clause.

Response: Correct the syntax of the DECLARE statement and re-execute the program.

RDX044E Expected keyword *expected* was not found – rather *unexpected* specified instead

Explanation: RDX encountered an unexpected keyword.

Response: Correct the unexpected token and re-execute the program.

RDX045E You attempted DECLARE cursor *cursor-name* that is already in use – issue CNTL RELEASE Cn

Explanation: The cursor Cn specified on a DECLARE statement is in use.

Response: To reuse cursor Cn, first release it using a RDX CNTL RELEASE Cn command. An open cursor may not be released. It must first be closed.

RDX046E You attempted PREPARE Cn that is already in use – issue CNTL RELEASE Sn

Explanation: The cursor Cn specified on a DECLARE statement is in use.

Response: To reuse cursor Cn, first release it using a RDX CNTL RELEASE Cn command. An open cursor may not be released. It must first be closed.

RDX047E Syntax: GET DIAGNOSTICS :string = ALL {STATEMENT | CONDITION | CONNECTION } {:instance} -- *token*

Explanation: The GET DIAGNOSTICS statement violates the syntax appearing in the message text. *token* is not correct.

Response: Correct the syntax of the GET DIAGNOSTICS statement and re-execute the program.

RDX048E Expected a host variable (:hv) but found *token*

Explanation: The statement syntax requires a host variable but *token* was encountered instead.

Response: Correct the SQL statement and re-execute it.

RDX049E Keyword *word* already detected – second occurrence is an error

Explanation: The keyword *word* cannot be repeated in this SQL statement.

Response: Correct the SQL statement and re-execute it.

RDX050E Statement terminated prematurely – review SQL statement syntax amd try again

Explanation: N/A

Response: You may have omitted a REXX continuation character to continue the SQL statement to subsequent lines.

RDX051E SQLDAID variable must be set to “SQLDA n” where n = blank, 3, or 3 -- *token*

Explanation: You attempted to create an SQLDA, but the variable stem.SQLDAID, (which is a required input variable) was not assigned or contains an invalid value.

Response: Assign a valid value to the SQLDAID variable as described in the message text.

RDX052E SQLD variable must be set to integer value 1 to 750 -- *token*

Explanation: You did not assign a value to the SQLD variable or the value was not correct.

Response: Assign a correct value to the SQLD variable and re-execute your program.

RDX053E Invalid SQLTYPE variable -- *token*

Explanation: See the DECLARE VARIABLE RDX statement for valid values of SQLTYPE.

Response: Correct the statement and re-execute your program.

RDX054E Invalid SQLLEN.SQLPRECISION - *token*

Explanation: You assigned an incorrect value of *token* to the REXX variable stem.SQLLEN.SQLPRECISION. A valid precision value is required for the DECIMAL SQLTYPE.

Response: Assign a valid value to stem.SQLLEN.SQLPRECISION and re-execute your program.

RDX055E Invalid SQLLEN.SQLSCALE - *token*

Explanation: You assigned an incorrect value of *token* to the REXX variable stem.SQLLEN.SQLSCALE.

Response: Assign a valid numeric value to the REXX variable stem.SQLLEN.SQLSCALE and re-execute your program.

RDX056E Invalid SQLLEN - *token*

Explanation: You attempted to construct an SQLDA from a REXX stem. The value *token* is not valid for the variable stem.SQLLEN

Response: Assign a valid numeric value to the REXX variable stem.SQLLEN and re-execute your program.

RDX057E Invalid SQLUSECCSID variable, not 0-32768, nor -1 -- instead *token* specified

Explanation: You attempted to construct an SQLDA from a REXX stem, but the value *token* is not valid for the variable stem.SQLUSECCSID. Only the values listed in the message are valid.

Response: Assign a valid numeric value to the REXX variable stem.SQLUSECCSID and re-execute your program.

RDX058E Invalid SQL verb *verbname*

Explanation: A token that follows EXECSQL in an ADDRESS RDX command is not a valid SQL verb.

Response: Review the specified verb, correct it, and re-execute your program.

RDX059E Syntax: CNTL {SNAP | RELEASE Sn} -- *token*

Explanation: You specified the invalid keyword *token* in a RDX CNTL command.

Response: Correct the syntax error and re-execute your program.

RDX060E DESCRIBE *Sn* was already done – second DESRIBE not allowed

Explanation: Only one DESCRIBE of an SQL statement *Sn* is permitted.

Response: Examine your program logic. Ensure that you do not issue more one DESCRIBE for the same prepared SQL statement.

RDX061E DESCRIBE INPUT was already done – second DESCRIBE INPUT not allowed

Explanation: Only one DESCRIBE INPUT for an SQL statement *Sn* is permitted.

Response: Examine your program logic. Ensure that you do not issue more than one DESCRIBE INPUT for the same prepared SQL statement.

RDX062E Requested SQL routine code *rtncode* is not valid for this version of Db2

Explanation: The SQL statement that you attempted to execute is not supported in this version of Db2. The *rtncode* is an internal routine code corresponding to the failed SQL statement.

Response: Review the SQL statement and make sure the Db2 version supports it.

RDX063E Number of input variables *varnum* not equal to number of SQLVARs *sqlvnum* – use either host vars or OPEN Cn USING DESCRIPTOR :stem

Explanation: The SQL OPEN Cn statement and its internal SQLDA contained several SQLVAR entries that are not equal to the number of host variables in the prepared SQL statement. Most likely, you executed PREPARE without a DESCRIBE for the statement *Sn* and attempted to OPEN Cn while the internal SQLDA was not yet built.

Response: If you constructed an SQLDA from a REXX stem, issue OPEN Cn USING DESCRIPTOR :stem.

RDX064E DATATYPE(nnn) invalid or non-numeric variable -- token

Explanation: The DECLARE VARIABLE statement contains a DATATYPE(nnn) keyword containing an invalid value *token*.

Response: Refer to the list of valid DATATYPE values defined by the syntax for the DECLARE VARIABLE statement.

RDX065E LENGTH(l)|LENGTH(p,s) parameter specifies non-numeric value *token*

Explanation: The DECLARE VARIABLE parameter LENGTH must contain a numeric value or values.

Response: Correct the LENGTH parameter.

RDX066E LENGTH(p,s) must be for DECIMAL datatype -- *token*

Explanation: DECLARE VARIABLE for a DECIMAL data type must specify a LENGTH(p,s) parameter.

Response: Correct the LENGTH parameter specification and re-execute your program.

RDX067E LENGTH(length) for *datatype* datatype must be value

Explanation: DECLARE VARIABLE for a *datatype* must specify a LENGTH parameter of *value*. For example, you specified DECLARE VARIABLE INTEGER LENGTH(8) but an INTEGER datatype must always have a length of 4.

Response: Correct the LENGTH parameter and re-execute your program.

RDX068E LENGTH(n) was not specified and there is no default length for *datatype* datatype

Explanation: The *datatype* data type has no default LENGTH attribute. Specify it explicitly.

Response: Specify the LENGTH parameter and re-execute your program.

RDX069E Variable name is limited to 30 characters – you specified *number*

Explanation: RDX limits the length of all variable names to 30 characters to make it compatible with the SQLNAME field of the SQLDA.

Response: Shorten the variable name to not exceed 30 characters.

RDX070E DATATYPE(sqltype) is a required parameter that was not specified

Explanation: You specified a DECLARE VARIABLE statement without an explicit data type (such as INTEGER, FLOAT, …). Nor did you specify a DATATYPE(sqltype) parameter.

Response: Correct the syntax error of the DECLARE VARIABLE statement and re-execute your program.

RDX071E Variable *varname* already defined – duplicate not allowed

Explanation: You issued a DECLARE VARIABLE request for *varname* more than once.

Response: Do not issue a duplicate DECLARE VARIABLE statement.

RDX072E Syntax: CALL :procname {(:hv, … | USING DESCRIPTIOR :stmt} – found *token*

Explanation: You violated the syntax of the CALL statement in the token *token.*

Response: Correct the syntax error and re-execute your program.

RDX074E Host variable *varname* was not declared

Explanation: The host variable *varname* must be declared with a DECLARE VARIABLE statement.

Response: Use DECLARE VARIABLE *varname* to declare the host variable used in the CALL statement.

RDX075E Host variable *varname* used as an indicator but was declared as a main variable

Explanation: The host variable varname was declared as a main variable but was used as an indicator variable. For example: CALL :proc (:mv:*varname*, …)

Response: Specify the INDICATOR keyword on a DECLARE VARIABLE statement.

RDX076E Syntax: ASSOCIATE {RESULT SET} LOCATOR(:loc) WITH PROCEDURE :procname – found *token*

Explanation: You violated the syntax of the ASSOCIATE statement in the token *token.*

Response: Correct the syntax error and re-execute your program.

RDX077E ALLOCATE Cn CURSOR FOR RESULT SET :rset, where n=101-200 – detected *token*

Explanation: You violated the syntax of the ALLOCATE statement in the token *token.*

Response: Correct the syntax error and re-execute your program.

RDX078E Invalid syntax CONNECT verb – error in the token token

Explanation: You violated the syntax of the CONNECT statement in the token *token.*

Response: Correct the syntax error and re-execute your program.

RDX080E REXX stem name is limited to 12 characters – you specified *charnum*

Explanation: RDX limits the length of stem names to 12 characters.

Response: Select a stem name that does not exceed 12 characters in length.

RDX081E The token *token* must be specified as :stem, where stem is a name of REXX stem used for SQLDA variables

Explanation: The *token* token must be the name of a REXX stem.

Response: Correct the syntax error and re-execute your program.

RDX082I You already successfully connected to Db2 subsystem – request ignored

Explanation: You issued a RDX INIT command when a Db2 connection was established.

Response: Revise your program to remove the redundant INIT call.

RDX083E Number of rows rows must be an integer n: 0<n<32767

Explanation: You specified a FOR :hv ROWS clause but the REXX variable hv contained an incorrect number of rows.

Response: Assign a valid integer number of rows to the hv variable and re-execute your program.

RDX084E Keyword *token* already recognized – duplication is not allowed

Explanation: The SQL statement contains the duplicate keyword *token.*

Response: Remove the duplicate keyword from the SQL statement and re-execute your program.

RDX085E SELECT INTO statement requires INTO clause that was not found

Explanation: The INTO clause is required in the SELECT statement.

Response: Specify an INTO clause in the SELECT statement and re-execute your program.

RDX086E SELECT INTO statement requires FROM clause that was not found

Explanation: The FROM clause is required in the SELECT statement.

Response: Specify a FROM clause in the SELECT statement and re-execute your program.

RDX087E Syntax: CNTL DSNTIAR {CALL | AUTO | NO} – not token

Explanation: The syntax of the CNTL statement was violated in the token *token*.

Response: Correct the statement syntax and re-execute your program.

RDX088E Statement *Sn* not used – CNTL RELEASE *Sn* failed

Explanation: You executed CNTL RELEASE for a statement that is not currently in use.

Response: Correct the error and re-execute your program.

RDX089E Statement Sn has an open cursor and may not be released

Explanation: You executed a CNTL RELEASE Sn request for the cursor Cn which is in an open state.

Response: Execute CLOSE Cn before issuing CNTL RELEASE Sn.

RDX090E You specified an INTO clause without a list of host variables

Explanation: Syntax error detected in the SELECT statement.

Response: Correct the INTO clause of the SELECT statement and re-execute your program.

RDX092E DECLARE CURSOR does not allow INTO clause

Explanation: Syntax error in the DECLARE CURSOR statement -- you can not specify an INTO clause. Instead, the INTO clause must be specified on the related FETCH statement. On the other hand, the DECLARE REXXSTEM service *requires* an INTO clause because the related FETCH is issued internally by RDX.

Response: Correct the syntax error and re-execute your program.

RDX093E Invalid statement syntax – error in token *token*

Explanation: Review the SQL statement syntax.

Response: Correct the syntax error in the SQL statement and re-execute your program.

RDX094E Statement S0 is invalid – logic error

Explanation: Program logic error.

Response: Report this problem to Broadcom Support.

RDX095E You attempted to PREPARE empty SQL statement

Explanation: You issued PREPARE FROM :stmt, where the REXX variable stmt is undefined or a null string.

Response: Ensure that the value of the REXX variable stmt is set with a valid SQL statement and re-execute your program.

RDX096E You specified DATATYPE keyword together with datatype name – this is not allowed

Explanation: You issued a DECLARE VARIABLE statement and specified the DATATYPE code and the explicit data type name. For example: DECLARE VARIABLE int INTEGER DATATYPE(496).

Response: Revise the SQL statement by removing the DATATYPE keyword or the explicit data type name.

RDX098E Parameter SYSTEM and PLAN can only be set before connection to Db2 is established

Explanation: The CNTL statement sets a default Db2 subsystem and plan name. This information is used for implicit connection to Db2 when an SQL call is issued in the absence of a previous INIT call. You issued a CNTL service with SYSTEM and PLAN parameters ***after*** the connection to Db2 was established. Doing so is not allowed.

Response: Delete this CNTL statement and re-execute your program.

RDX099E Failed create Name-Token, RC=*retcode*

Explanation: Program logic error.

Response: Report this problem to Broadcom Support.

RDX100E Failed to connect to Db2 subsystem *ssid* with plan *planname* – but you continue invoking RDX

Explanation: You attempted to execute an SQL statement even though the previous Db2 connection attempt failed.

Response: Revise your program logic to check the SQLCODE or RC after connecting to Db2. If the connection to Db2 fails, abort program execution or try to connect to a different Db2 subsystem.

RDX101E Failed to convert result set locator value to EBCDIC from BINARY, RC=*retcode*

Explanation: Program logic error.

Response: Report this problem to Broadcom Support.

RDX102E *calltype* ended with RC = *retcode*

Explanation: You issued a COMMIT or ROLLBACK request of type CPIC which failed with the return code *retcode*. The *calltype* is SRRCMIT or SRRBACK.

Response: See the IBM publication ‘MVS Callable Services for HLL’ for a description of SRRBACK and SRRCMIT and the return codes from these services.

RDX103E Variable *varname* was not defined

Explanation: You did not define the REXX variable named *varname* but it was used as a host variable in the SQL statement.

Response: Revise your program to assign a value to the REXX variable *varname,* and re-execute your program.

RDX104E Variable *varname* must be of a numeric data type – not *token*

Explanation: The REXX variable *varname* must be of a numeric data type, but contains value of *token*.

Response: Ensure that the REXX variable *varname* is assigned a numeric value, and re-execute your program.

RDX105E The xLOB_FILE variable *varname* was not defined –must be set with a dataset name of existing VB file

Explanation: The REXX variable *varname* is used as a xLOB_FILE data type, so it must contain the name of an existing dataset with RECFM(VB).

Response: Ensure that the variable *varname* is assigned a correct value and re-execute your program.

RDX106E Invalid syntax of a special SET statements for scalar functions LENGTH, SUBSTR, POSSTR, CONCAT – *token*

Explanation: A syntax error was detected in the special form of the SET statement – the *token* is the culprit. Review the section entitled ***SET statements for LOB LOCATOR processing*** in this publication. RDX has limited support for the SET statement that is strictly enforced. Specify these SET statements exactly as documented.

Response: Revise your program to ensure that the SET statement strictly adheres to the RDX syntax specification.

RDX107E Syntax: CNTL ERRORS {RETURN | CANCEL} - &token

Explanation: A syntax error was detected in the CNTL ERRORS statement – the &*token* is the culprit. Use correct keywords RETURN or CANCEL.

Response: Correct CNTL statement and re-execute failed exec.

RDX108E The command &command was specified without any parameters

Explanation: Invalid syntax of *&command* command.

Response: Correct syntax error of *&command* and re-execute exec.

RDX109E SQL statement text length exceeded specified maximum of *&max* characters in RDX$TSD

Explanation: Review the maximum LOB data type size in RDX$TSD defaults CSECT, RDXMAXL# field. SQL statement is of the CLOB type. Default RDXMAXL# = 262144. This value is only limited by the amount of extended region of your address space.

Response: Review the SQL statement length and increase RDXMAXL# so that it exceeds the SQL statement.

RDX110E Syntax: CNTL DEBUG {SQL | APPLICATION | ISPF | MEMORY|ALL|NONE -- *&token*

Explanation: Review CNTL command syntax and correct failed statement.

Response: Review the SQL statement and re-execute exec.

RDX111D *&debug*

Explanation: Generic debug message of variable content.

Response: If you suspect an error, report it to Broadcom Support.

RDX112I Command interrupted. Clear the screen and press the ENTER key

Explanation: Attention key was pressed (PA2). Follow the on-screen instructions.

Response: Follow the on-screen instructions.

RDX113I Connect to the inactive Db2 &SSID which was terminated by attention

Explanation: Attention key was pressed (PA2). Follow the on-screen instructions.

Response: Follow the on-screen instructions.

RDX114I RLX interrupted. Select from one of the following choices: RDX115I No RLX statement was executing when you signaled attention: RDX116W Wrong reply, RLX command will continue ... RDX117W OK, trying to terminate RLX ... RDX118W OK, trying to halt REXX exec ... RDX119W OK, resuming execution ... RDX120W OK, issuing user abend ... RDX121W OK, trying to SNAP RLX control blocks ...

Explanation: Attention key was pressed (PA2). Follow the on-screen instructions. All these messages may be issued based on your response to a prompt.

Response: Follow the on-screen instructions.

RDX122E ISPFTABLE service may only be invoked in ISPF environment

Explanation: You issued DECLARE Cn ISPFTABLE statement why not active in TSO/ISPF.

Response: Execute failed exec in TSO/ISPF environment.

RDX123D SERVICE(&&S) CSECT(&&C) OFFSET(&&O) RC(&&RC)

Explanation: Debugging messages that assist Broadcom Support when diagnosing a problem.

Response: Report this message to Broadcom Support.

RDX124W ISPF is not active - service failed

Explanation: You executed REXX exec outside of TSO/ISPF.

Response: Review your exec invocation environment.

RDX125E Cursor or statement name must not exceed 30 characters in length - *&cursor*

Explanation: You used *&cursor* name greater than 30 characters. This value is not allowed.

Response: Review and revise *&cursor* name.

RDX126E ISPF table name must not exceed 8 characters in length - *&table*

Explanation: Specified ISPF table name that is used in DECLARE Cn ISPFTABLE service must be no greater than eight characters.

Response: Review and revise *&table* name.

RDX127E All cursor/statement slots are used, execute CNTL RELEASE *&cursor*, then re-execute the statement

Explanation: In your active exec, you used more that 100 cursors. Execution aborted.

Response: Review your DECLARE Cn SQL statements and make sure you make use of host variables (for example, COLN = :hostvar) rather than substituted REXX variables (for example, “COLN=“ hostvar) to limit number of unique cursor-based SQL statements. If such revision is not possible, use **CNTL RELEASE &cursor** statement.

RDX128E ISPFTABLE service cannot be used for a result set with more than 254 columns

Explanation: ISPF limits number of keyed and non-keyed variables in an ISPF table to 254.

Response: This error may occur when you use DECLARE tabname FOR SELECT * and the result set is greater than 254 columns. Revise SELECT statement to specify only columns you need to fetch and limit it number to 254.

RDX129E Syntax: CNTL ISPFTABLE {EXTEND|DELETE} - *&token*

Explanation: You made an error in the CNTL ISPFTABLE statement.

Response: Correct the error and re-execute.

RDX130E Service *&service* RC=*&RC* -- logic error

Explanation: RDX internal error.

Response: Report this error to Broadcom Support.

RDX131E ISPF table *&table* has *&number1* columns while result set has *&number2* and you used EXTEND function

Explanation: Review failed statement and related DECLARE table ISPFTABLE statement. Compare number of columns in the result set and in the created ISPF table.

Response: Correct the problem and re-execute.

RDX132E ISPF table &table has column names that differ from the INTO clause of DECLARE statement

Explanation: You might have used a predefined ISPF table with the DECLARE table ISPFTABLE statement.

Response: Review ISPF table and DECLARE table ISPFTABLE statement.

RDX133E DECLARE table ISPFTABLE service INTO clause varnames must be =< 8 chars in length with no underscore chars

Explanation: Column names in the INTO class of ISPFTABLE service cannot be greater than eight characters in length and cannot contain no underscore character “_”.

Response: Such column names are an ISPF requirement. Change the INTO clause.

RDX134E SQLDAID variable must be set to "SQLDA 2" when LOB columns are present

Explanation: You constructed an input SQLDA in your exec and specified one or more column datatypes as LOB.

Response: Make sure that you specified SQLDAID as “SQLDA 2” or “SQLDA 3” if labels are used.

RDX135E Syntax: EPANEL DISPLAY|NONDISPL - error in *&token*

Explanation: You made an error in the CNTL EPANEL statement.

Response: Correct the error command and re-execute.

RDX136E DSNTIAR CALL is only allowed when DSNTIAR NO is in effect - command ignored

Explanation: You attempted to directly invoke DSNTIAR while DSNTIAR is in effect and loaded by RDX. DSNTIAR is used internally.

Response: Specify CNTL DSNTIAR NO or do not invoke DSNTIAR directly.

RDX137E In FOR *n* ROWS, n must be an integer not - &token

Explanation: Review and revise value of n in the FOR n ROWS clause. Confirm that *n* is a whole number 1 through 32768.

Response: Correct your exec and re-execute.

RDX138D Cursor <cursor> was auto-released due to COMPAT=Y

Explanation: The **CNTL COMPAT Y** option was specified during exec's execution or as a default parameter. Cursor <cursor> was released automatically after it was closed.

Response: N/A

RDX139D COMPAT=Y mode was activated

Explanation: Command **CNTL COMPAT Y** was issued.

Response: N/A

RDX140D COMPAT=N mode was inactivated

Explanation: Command CNTL COMPAT N was issued.

Response: N/A

RDX141E Detected potential end of memory - review columns data types, lengths and region size

Explanation: The SQL statement results in memory allocation that may cause SX78 abend. RDX rejects this SQL statement and issues this error message.

Response: Review column data types and their lengths. Compute total necessary storage to be allocated to hold a required result set. Compare required memory size with a region size of your address space. Increase the region size or change your SQL statement so that a required result set fits in an available storage. Re-execute your exec.

Coding Techniques and Diagnostic Procedures

This section provides guidelines for coding REXX programs that contain RDX commands and describes how to diagnose and correct errors in REXX, RDX, or both.

When developing REXX programs to interface with Db2, mistakes happen. These mistakes require analysis so you can understand and correct the problem. The errors that you may encounter can be grouped into three classes: REXX errors, errors that are detected by RDX as a result of command syntax violations and errors that are returned by Db2.

Handling REXX errors

REXX errors are due to REXX syntax violations and can be diagnosed with a REXX TRACE ‘opt’ statement. Usually, use TRACE ‘R’ to see the results of symbolic substitution of REXX statements and commands. For more details, use the TRACE ‘I’ command.

If you receive RC = -3, this value means in REXX terminology, ‘*command not defined*’. This message happens when you do not establish a TSO/ISPF REXX environment to execute ADDRESS RDX commands. To review the steps of RDX command enablement, see RDX installation articles.

Handling RDX Errors

RDX command errors are due to syntax violations of the RDX implementation of Db2 SQL. RDX detects these errors before the command is passed to Db2. You know of the error by examining the SQLCODE after the SQL statement executes. The SQLCODEs that correspond to RDX triggered errors have the format -100nnn, where nnn is a RDX error message number. The RC variable is also set with the value of nnn. After the RDX INIT command, the REASON variable is also set with the reason code that is returned by the Db2 attachment facility (CAF or RRSAF). The REXX stem SQLERR.i contains the RDX error message text, which you can display to diagnose an error.

```
"EXECSQL SELECT NAME FROM SYSIBM.SYSTABLES",
 "FETCH FIRST 1 ROWS ONLY"
```

This statement results in the SQLCODE variable being set to -100085, which means that an RDB085E message was issued. The RC variable is set to -1. You can display the error message using the following REXX statements:

```
DO i = 1 TO sqlerr.0
   SAY sqlerr.i
END
```

In this case, the following message is displayed:

```
RDB085E - SELECT INTO statement requires INTO clause that was not found
```

This message indicates that you specified a SELECT INTO statement. You did not include an INTO clause. The following sample illustrates how to specify a SELECT INTO statement correctly:

```
"EXECSQL SELECT NAME INTO :NAME FROM SYSIBM.SYSTABLES",
 "FETCH FIRST 1 ROWS ONLY"
```

Handling Db2/SQL Errors

Db2/SQL error codes are returned by Db2 and are documented in the IBM Db2 publications: *Db2 Vx for z/OS Messages* and *Db2 for z/OS Codes*. After the SQL statement is executed, the REXX stem variable SQLERR.i contains the error message.

The following statement specifies an incorrect name for a Db2 catalog table:

```
"EXECSQL SELECT NAME INTO :NAME FROM SYSIBM.XYSTABLES"
```

The SQLCODE code is set to -204 and the RC variable is set to -1. The SQLERR.i stem contains the following multi-line message text:

```
DSNT408I SQLCODE = -204, ERROR:  SYSIBM.XYSTABLES IS AN UNDEFINED NAME
DSNT418I SQLSTATE   = 42704 SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNXOTL SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD    = -500  0  0  -1  0  0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD    = X'FFFFFE0C'  X'00000000'  X'00000000'  X'FFFFFFFF'
         X'00000000'  X'00000000' SQL DIAGNOSTIC INFORMATION
```

You may want to see a trace of the actual SQL statements executed. Activate the SQL monitor trace with the following parameters on a RDX INIT command:

```
"INIT SYSTEM(DSN) SQLMON(FULL) SQLERROR"
```

When you execute a REXX exec containing RDX commands, you can find a trace similar to the following code on the job output queue for your TSO session or written to DDNAME USQPRINT of a batch job:

```
= Relational Data Interface (RDI) Trace             PAGE      1
-------------------------------------------------------------------------------------------------------------
= PROG     Stmt# Sec# Version        SQL Type   P# Flag Ctyp Vparm    AuxParm  CPU Time  Elapsed Time
-------------------------------------------------------------------------------------------------------------
= RDBSQL9   772  100 180BD4640DE42D40 PREPARE   64 4400 0023 000A95B8 0B154840 00.026636 00:00.106940
=          SQLDA    SIZE(104   ) DIM(2   ) USED(2   )
=           SQLVARN TYPE(01C1) LENGTH(00FE) HVNAME() IV(FFFF)
=              DATA
=           SQLVARN TYPE(01C0) LENGTH(7FFF) HVNAME() IV(N/A)
=              DATA SELECT NAME FROM SYSIBM.XYSTABLES
DSNT408I SQLCODE = -204, ERROR:  SYSIBM.XYSTABLES IS AN UNDEFINED NAME
DSNT418I SQLSTATE   = 42704 SQLSTATE RETURN CODE
DSNT415I SQLERRP    = DSNXOTL SQL PROCEDURE DETECTING ERROR
DSNT416I SQLERRD    = -500  0  0  -1  0  0 SQL DIAGNOSTIC INFORMATION
DSNT416I SQLERRD    = X'FFFFFE0C'  X'00000000'  X'00000000'  X'FFFFFFFF'
         X'00000000'  X'00000000' SQL DIAGNOSTIC INFORMATION

-------------------------------------------------------------------------------
=SQL Summary Report
-------------------------------------------------------------------------------
DBRM/PGK STMT# Statement description    Calls# CPU Time        Elapsed Time
RDBSQL9    772 PREPARE                       1 00:00:00.026636 00:00:00.106940
```

Another use of this report is as a performance monitoring tool that allows you to identify performance bottlenecks and see details of RDX internal processing. The report can help you redesign your SQL statements to improve application performance.

Displaying All REXX Variables Defined in Your Program

During program development, you may want to see all variables defined in your program. This information may be especially important after calling interface programs, such as RDX, which creates REXX variables to communicate with an invoked REXX program. The sample program in the RDXSMPVD member of the CRAIEXEC library can be used to display all defined REXX variables. RDXSMPVD invokes the RDX function REXXVARS to retrieve a list of the names of all defined REXX variables. RDXSMPVD then displays each variable name and its value.

This example displays all REXX variables:

```
/* REXX */
ADDRESS RDX
"INIT SYSTEM(DSN)"
"EXECSQL SELECT NAME INTO :NAME FROM SYSIBM.SYSTABLES",
 "FETCH FIRST 1 ROWS ONLY"
CALL DUMPVARS
"TERM"
RETURN 0

DUMPVARS:
 $V                = REXXVARS()
 DO i              = 1 TO WORDS($V)
    var            = WORD($V,i)
    SAY LEFT(var,30,'.') val
 END
RETURN 0
```

This REXX program produces the following output:

```
RC............................. 0
SIGL........................... 6
NAME........................... VACT
Db2SSN......................... DB9A
REASON......................... 00000000
Db2VRM......................... 910
SQLCODE........................ 0
SQLERRP........................ DSN
SQLERR.0....................... 1
SQLERR.1....................... DSNT400I SQLCODE = 000,  SUCCESSFUL EXECUTION
SQLERRD.1...................... 0
SQLERRD.3...................... 0
SQLERRD.2...................... 0
SQLERRD.6...................... 0
SQLERRD.5...................... 0
SQLERRD.4...................... -1
SQLSTATE....................... 00000
SQLWARN.0......................
SQLWARN.2......................
SQLWARN.1......................
SQLWARN.4......................
SQLWARN.6......................
SQLWARN.5......................
SQLWARN.8......................
SQLWARN.10.....................
SQLWARN.9......................
SQLWARN.7......................
SQLWARN.3......................
SQLERRMC.......................
```

RDX Sample Programs

RDX provides an extensive set of sample programs. These sample programs illustrate the execution of SQL statements, different program development methods, and various RDX REXX functions that may be useful in program development. All RDX sample programs use the naming pattern RDXSMPxx, where xx is a code identifying a specific sample program. This article describes RDX sample programs.

Sample name	Description
RDXSMPID	Provides initialization defaults for all the sample programs. You specify the Db2 subsystem name and define the RDX REXX host command environment in this subroutine.
RDXSMPIL	Illustrates the use of the SET CURRENT PACKAGESET statement to select an isolation level (either CS, UR, RR, or RS) for the SQL statements to be executed in a REXX program. The default isolation level is CS.
RDXSMPP1	Sample stored procedure 1. The DDL to create this stored procedure is found in the RDXSDDL member of the RDXCNTL library. It is used in other RDX sample programs.
RDXSMPSE	A fragment of REXX code that can be copied into your programs. It defines RDX as a REXX Host Command.
RDXSMPVD	A REXX subroutine that can be copied into your REXX program. When you call this subroutine, all REXX variables that are defined in your REXX program is displayed, with their values. You can use this subroutine for problem diagnosis.
RDXSMP01	Illustrates implicit connection to a Db2 subsystem. In order to use this program, you must modify line 000014 to specify the name of a Db2 subsystem in which RDX is installed. RDXSMP01 does not issue a RDX INIT command. Rather, a CNTL command is used to configure RDX with the desired Db2 subsystem ID, before the first SQL call is issued. RDX automatically connects to the specified Db2 subsystem.
RDXSMP02	Illustrates usage of the INTO SQLNAME clause of the SELECT INTO statement, with the usage of the CNTL PREFIX command.
RDXSMP03	Illustrates invocation of RDX as a REXX function, rather than as a REXX Host Command. These two methods are functionally equivalent and can be selected and used interchangeably. REXX Host Command usage may be easier to debug with a REXX TRACE ‘c’ statement.
RDXSMP04	Illustrates SQL CONNECT statements. To successfully execute this program, install RDX in more than one Db2 subsystem and change lines 000023-000025 to specify the location name of a remote Db2 subsystem, user ID, and password.
RDXSMP05	Provides an example of the CALL statement and the action that is required for its successful execution. To successfully execute this program, create the RDXSMPP1 REXX stored procedure (see the RDXSMPP1 sample) and install it in WLM stored procedure address space. You must also add the CRAILRAI and CRAILOAD libraries to the STEPLIB concatenation of the JCL for the WLM stored procedure address space if these libraries are not already in the LINKLIST. The RDXDEMO table must also exist (see the DDL in the RDXCNTL library member RDXSDDL).

Sample name	Description
RDXSMP06	Create the RDXDEMO table that is used in the sample programs and populate it with test data. This program illustrates the SQL CREATE TABLE statement as issued from a REXX program. Note that the sample table RDXDEMO is also created by the DDL in the RDXSDDL member of RDXCNTL library.
RDXSMP07	Illustrates the following SQL EXECUTE IMMEDIATE statements: • ALTER RDXDEMO table • Create a COMMENT for the RDXDEMO table • Create a GLOBAL TEMPORARY TABLE
RDXSMP08	Provides an example of a CALL statement with the USING DESCRIPTOR clause. This is alternative usage for the SQL CALL statement (See RDXSMP05 for a comparison).
RDXSMP09	Illustrates the SQL COMMIT, ROLLBACK, SAVEPOINT, and RELEASE SAVEPOINT statements.
RDXSMP10	Illustrates the SQL PREPARE and DESCRIBE statements.
RDXSMP11	Illustrates the SQL INSERT, UPDATE, and DELETE statements
RDXSMP12	Illustrates the SQL ALTER CLONE, EXCHANGE, and TRUNCATE statements introduced in Db2 V9.
RDXSMP13	Illustrates the DECLARE REXXSTEM service. The RDXDEMO table is required by this sample program. RDXSMP13 illustrates three forms of the INTO clause on the SELECT statement.
RDXSMP14	Compares and contrasts two methods of cursor processing: • A DECLARE CURSOR statement with OPEN, FETCH, and CLOSE statements vs. • A PREPARE, DESCRIBE, DESCRIBE INPUT with OPEN, FETCH, and CLOSE statements.
RDXSMP15	Illustrates the SQL EXECUTE statement: • First we PREPARE, DESCRIBE INPUT, and EXECUTE an UPDATE statement that uses input host variables • The commented statements illustrate how an input SQLDA could have been manually created and used in an EXECUTE USING DESCRIPTOR statement • A SQL statement without input host variables is passed to EXECUTE IMMEDIATE.
RDXSMP16	Illustrates the SQL FETCH statement with the ROWSET POSITIONING and FOR :rows ROWS clauses that are introduced in Db2 V8.
RDXSMP17	Illustrates working with LOB data – using the LOB_FILE data type. In order to use this sample program, you must create a RDX.RDXLOBS table using the DDL found in the RDXSLOB member of the RDXCNTL library.
RDXSMP18	Illustrates working with LOB data – using LOB locators. In order to use this sample program, you must create a RDX.RDXLOBS table using the DDL found in the RDXSLOB member of the RDXCNTL library.
RDXSMP19	Illustrates working with LOB data – using LOB data types. In order to use this sample program, you must create a RDX.RDXLOBS table using the DDL found in the RDXSLOB member of the RDXCNTL library.

Sample name	Description
RDXSMP20	Illustrates the SQL MERGE statement that is introduced in Db2 V9. Note that this sample program creates a clone of the RDXDEMO table RDXDEMO_AR and then truncates it – thus the contents of the RDXDEMO table is lost. You may need to run the RDXSMP06 sample program to re-create the RDXDEMO table.
RDXSMP21	Illustrates the pureXML support that is introduced in Db2 9. This sample program creates a table that is named MYCUSTOMER with an XML column. The MYCUSTOMER table is created for each individual user that executes this program. You may need to DROP this table later in order to clean up.
RDXSMP22	Illustrates the use of XMLQUERY to select part of an XML document. RDXSMP22 illustrates the use of host variables in SQL/XML and shows a SQLMON trace. This sample program uses the MYCUSTOMER table that is created by the RDXSMP21 program.
RDXSMP23	Demonstrates the Db2 IFI calls: COMMAND and READS.
RDXSMP24	Demonstrates two ways to DECLARE Cn CURSOR WITH HOLD statement. First as (a) DECLARE and (b) as PREPARE, DESCRIBE, and DESCRIBE INPUT statements. The form (a) does what (b) but under the covers, as one composite operation.
RDXSMP25	Demonstrates execution of DECLARE Cn REXXSTEM statement with INTO SQLNAME and INTO SQLNAMEO clauses. Shows difference in names of output stems.
RDXSMP26	Demonstrates DECLARE Cn ISPFTABLE service and RDXDTAB exec to display any ISPF table.
RDXSMP27	Demonstrates multi-row FETCH SQL statements.
RDXSMP28	Demonstrates multi-row INSERT SQL statements.
RDXSPEXP	This value is a REXX version of the C language procedure DSN8EXP that performs SQL EXPLAIN processing.
RDXSQLDA	This utility REXX program displays the SQLDA mapped into a REXX stem variable. RDXSQLDA provides a means to display the SQLDA associated with a specific SQL statement. The sample programs RDXSMP05 and RDXSMP08 illustrate how to invoke this program. RDXSQLDA uses the REXX GLOBVAR function. Thus, it must be invoked with the SDWB front-end program.

Before you invoke the sample programs, follow these steps:

- Copy the CRAIEXEC library into your own SYSEXEC or SYSPROC library
- Ensure that the RDXSMPID sample program specifies the Db2 subsystem ID where RDX is installed.
- Review the source code of a sample REXX exec and make sure that all prerequisites for the program execution are met, For example. the RDXDEMO table and RDXSMPP1 stored procedure are both defined.

You can invoke a RDX sample program from Option 6 of the TSO/ISPF environment.

Conversion of MAX/REXX Execs to RDX

This section describes considerations and programming effort necessary to convert REXX execs that are written using the MAX/REXX Db2 interface to the RDX SQL interface. RDX provides a REXX function RXSQL that is compatible with the MAX/REXX function that performs the command mappings that are documented in the following table.

MAX/REXX statement in rx = RXSQL(*command*)	Converted to RDX statement
'CONNECT ssid'	'INIT SYSTEM(ssid)'
'DISCONNECT'	'TERM'
'CNTL'	Pass through without change
'INIT ...'	Pass through without change
'TERM ...'	Pass through without change
any-other-command	'EXECSQL' any-other-command

Handling the MAX/REXX 'CONNECT' Statement

The 'CONNECT' statement is not supported by RDX because it conflicts with the DB2 SQL CONNECT statement, thus it is translated by the RXSQL function to a RDX 'INIT' statement. For example, if your REXX exec specifies:

```
$rc = RXSQL('CONNECT Db2T')
```

then this statement is converted to an:

```
'INIT SYSTEM(Db2T)'
```

command and is passed to RDX which recognizes it as a valid RDX statement. Note that RDX error handling after the CONNECT statement will be different. For example:

```
$rc = RXSQL('CONNECT Db2T')
IF $rc <> 0 THEN DO
    SAY 'CONNECT ERROR - RC='rc 'REASON='reason
    EXIT 8
END
```

The values that are returned in the RC and REASON variables are documented in the IBM Db2 for z/OS Messages and Codes publication and allows you to identify the reason why the exec failed to connect to Db2.

Since the MAX/REXX CONNECT statement does not allow you to specify the name of the Db2 application plan, make sure the default plan name that is specified at RDX installation time is correct. The job resided in hlq.RDXCNTL(RDXJTSD) allows the RDX installer to change RDX defaults. For example, to change the default application plan, edit and submit the RDXJTSD job specifying:

```
//DEFAULT EXEC PGM=IRXJCL,
// PARM='RDXJTSD PLAN(RLX916CS)'
```

> **NOTE**
> To avoid confusing when specifying the Db2 SQL 'CONNECT …' statement and RXSQL 'CONNECT ssid' statement, always specify SQL commands with the EXECSQL prefix, for example, 'EXECSQL sql_command …'

Handling the MAX/REXX ‘DISCONNECT’ Statement

RDX translates the RXSQL ‘DISCONNECT’ statement to a RDX ‘TERM’ statement. To use the full capabilities of the RDX ‘TERM’ statement, specify it directly, for example:

```
$rc = RXSQL('TERM ABRT')
```

Handling MAX/REXX SQL Statements

Any SQL statements that are invoked in RXSQL function calls, other than those with special handling as described in Table 1, are considered SQL statements and is prefixed with an ‘EXECSQL’ string before being presented to RDX as valid SQL statements. However there are certain incompatibilities between the specification of MAX/REXX and RDX SQL statements which are discussed individually in the following sections.

Handling MAX/REXX SELECT INTO STEM() Statements

The following examples of the MAX/REXX SELECT INTO STEM() statement and the RDX DECLARE REXXSTEM statement highlight the differences between them and outlines the manual conversion steps required.

Sample code for a MAX/REXX SELECT INTO STEM() statement follows.

```
CALL "RXSQL" "SELECT NAME, DBNAME, TSNAME",   (1)
      "INTO STEM()",
       "FROM SYSIBM.SYSTABLES",
        "ORDER BY DBNAME, TSNAME, NAME"
```

This statement returns REXX stemmed variables as follows:

- **DBNAME.0** - is the number of returned rows
- **DBNAME.i** - i-th row of DBNAME column
- **TSNAME.i** - i-th row of TSNAME column
- **NAME.i** - i-th row of NAME column

Compare this information with the RDX DECLARE REXXSTEM statement:

```
"EXECSQL DECLARE C2 REXXSTEM FOR",           (2)
  "SELECT NAME, DBNAME, TSNAME",
   "INTO :name, :dbname, :tsname",
    "FROM SYSIBM.SYSTABLES",
     "ORDER BY DBNAME, TSNAME, NAME"
```

The SQL statement returns the following REXX stemmed variables:

- **C2.ROWS** - is the number of rows that are returned by cursor C2
- **DBNAME.i** - i-th row of DBNAME column
- **TSNAME.i** - i-th row of TSNAME column
- **NAME.i** - i-th row of NAME column

If a table column is declared without a ‘NOT NULL’ clause, RDX implicitly creates an indicator variable for that column of the form: varname_.i. That is, RDX appends an underscore character at the end of the variable name. MAX/REXX instead, creates an indicator variable by appending a question mark ‘?’ at the end of the variable name. In this example, RDX created a series of REXX stemmed variables, each of which contains C2.ROWS, where C2 is the cursor name.

The manual conversion of statement (1) to statement (2) may be accomplished using the following steps:

- Insert before the SELECT statement in (1) the string **"EXECSQL DECLARE Cx REXXSTEM FOR"**. Specify a proper cursor name, which must be unique in the given exec. You can assign this string to a variable and then just insert it in front of the SELECT.

- Revise the INTO clause in the following way:
  - Delete the STEM() string
  - In the INTO clause, specify the names of the variables to correspond to the columns as a list of host variables
- Ensure the exec processes the number of rows that are contained in Cx.ROWS for each stemmed variable.
- Change each occurrence of the ‘?.’ string to the ‘_.’ string to convert MAX/REXX indicator variable names to RDX format.

Handling MAX/REXX SELECT INTO ISPTABLE(name) Statements

The following examples of the MAX/REXX SELECT INTO ISPTABLE(name) statement and the RDX DECLARE ISPFTABLE statement highlight the differences between them and describes the manual conversion steps required.

Sample code for a MAX/REXX SELECT INTO ISPTABLE(name) follows:

```
CALL "RXSQL" "SELECT NAME, DBNAME, TSNAME",  (1)
 "INTO ISPTABLE(TBLNAME)",
  "FROM SYSIBM.SYSTABLES",
   "ORDER BY DBNAME, TSNAME, NAME"
```

An example of the corresponding RDX DECLARE ISPFTABLE statement follows:

```
CALL "RXSQL" "DECLARE TBLNAME ISPFTABLE FOR",(2)
 "SELECT NAME, DBNAME, TSNAME",
  "FROM SYSIBM.SYSTABLES",
   "ORDER BY DBNAME, TSNAME, NAME"
```

To convert statement (1) to statement (2), insert the string “DECLARE TBLNAME ISPFTABLE FOR” and delete the “INTO ISPTABLE(TBLNAME) clause. In both cases, the ISPF table that is created can be accessed with the following block of REXX code:

```
ADDRESS ISPEXEC
cn = 'TBLNAME'
"TBTOP" cn
"TBSKIP" cn
DO WHILE rc = 0
   SAY LEFT(name,18),
       LEFT(dbname,10),
       LEFT(tsname,10)
   "TBSKIP" cn
END
```

The exec code that references the ISPF table TBLNAME need not be changed since the table has the same name and structure in both case (1) and case (2).

Documentation Legal Notice

This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the “Documentation”) is for your informational purposes only and is subject to change or withdrawal by Broadcom at any time. This Documentation is proprietary information of Broadcom and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of Broadcom.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all Broadcom copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to Broadcom that all copies and partial copies of the Documentation have been returned to Broadcom or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, BROADCOM PROVIDES THIS DOCUMENTATION “AS IS” WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL BROADCOM BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF BROADCOM IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is Broadcom Inc.

Provided with “Restricted Rights.” Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.


BROADCOM®
MAINFRAME SOFTWARE