Rocket | M204

Rocket Model 204 User Language/ DATABASE2 Interface Guide

Version 7 Release 4.0

May 2012
204-74-DB2-01

Rocket

m204.rocketsoftware.com

Notices

Edition

Publication date: May 2012

Book number: 204-74-DB2-01

Product version: Rocket Model 204 User Language/DATABASE2 Interface Guide -

Version 7 Release 4.0

Copyright

Trademarks

License agreement

Contact information

Web Site: www.rocketsoftware.com

Rocket Software, Inc. Headquarters
77 4th Avenue, Suite 100
Waltham, MA 02451–1468
USA
Tel: +1.617.614.4321
Fax: +1.617.630.7100

Contacting Technical Support

If you have current support and maintenance agreements with Rocket Software and CCA, contact Rocket Software Technical support by email or by telephone:

Email: m204support@rocketsoftware.com

Telephone :

North America	+1.800.755.4222
United Kingdom/Europe	+44 (0) 20 8867 6153

Alternatively, you can access the Rocket Customer Portal and report a problem, download an update, or read answers to FAQs. You will be prompted to log in with the credentials supplied as part of your product maintenance agreement.

To log in to the Rocket Customer Portal, go to:

http://www.rocketsoftware.com/support

Contents

About this Guide

The User Language/DATABASE 2 Interface (UL/DB2) provides a multithreaded connection from a Model 204 User Language procedure to a DB2 database to retrieve DB2 data or manipulate DB2 tables.

This guide describes:

- UL/DB2 Interface program environment
- Preparing a Model 204 environment to support the interface
- SQL statements supported by the interface, and how to include these statements in a User Language procedure
- Messages returned by the interface

Installation

This guide does not describe how to install UL/DB2. UL/DB2 is installed using INS204, the Model 204 automated installation facility. See the *Model 204 z/OS Installation Guide* for more information on installing UL/DB2.

Audience

This guide addresses three audiences:

Audience	Responsible for...
Model 204 system managers	Making a Model 204 environment ready to support the interface, and defining and controlling security. System managers should know how to set up Model 204 Online and BATCH204 environments.
DB2 database administrators	Granting PLAN and table privileges. DB2 database administrators should know how to precompile and bind a PLAN, and grant access to a PLAN.
Application programmers	Developing User Language procedures that use embedded SQL statements. Application programmers should know User Language and SQL syntax.

Model 204 documentation set

The complete commercially released documentation for the latest version of Model 204 is available for download from the Rocket M204 customer portal.

To access the Rocket Model 204 documentation:

1. Navigate to:

   http://www.rocketsoftware.com/m204

2. From the drop-down menu, select **Products > Model 204 > Documentation**.

3. Click the link to the current release and select the document you want from the list.

4. Click the .zip file containing the document.

5. Choose whether to open or save the document:
   - Select **Open** and double-click the pdf file to open the document.
   - Select **Save as** and select a location to save the zip file to.

Documentation conventions

This guide uses the following standard notation conventions in statement syntax and examples:

Convention	Description
`TABLE`	Uppercase represents a keyword that you must enter exactly as shown.
`TABLE` *`tablename`*	In text, italics are used for variables and for emphasis. In examples, italics denote a variable value that you must supply. In this example, you must supply a value for *tablename*.
`READ [SCREEN]`	Square brackets ([ ]) enclose an optional argument or portion of an argument. In this case, specify READ or READ SCREEN.
`UNIQUE | PRIMARY KEY`	A vertical bar (|) separates alternative options. In this example, specify either UNIQUE or PRIMARY KEY.
`TRUST |` <u>`NOTRUST`</u>	Underlining indicates the default. In this example, NOTRUST is the default.
`IS {NOT | LIKE}`	Braces ({ }) indicate that one of the enclosed alternatives is required. In this example, you must specify either IS NOT or IS LIKE.
`item ...`	An ellipsis (. . .) indicates that you can repeat the preceding item.
`item ,...`	An ellipsis preceded by a comma indicates that a comma is required to separate repeated items.
`All other symbols`	In syntax, all other symbols (such as parentheses) are literal syntactic elements and must appear as shown.
`nested-key` `::=` *`column_name`*	A double colon followed by an equal sign indicates an equivalence. In this case, *nested-key* is equivalent to *column_name*.

Convention	Description
`Enter your account:` `sales11`	In examples that include both system-supplied and user-entered text, or system prompts and user commands, boldface indicates what you enter. In this example, the system prompts for an account and the user enters **sales11**.
File > Save As	A right angle bracket (>) identifies the sequence of actions that you perform to select a command from a pull-down menu. In this example, select the Save As command from the File menu.
EDIT	Partial bolding indicates a usable abbreviation, such as E for EDIT in this example.

1

User Language/DATABASE 2 Program Environment

In this chapter

- Overview
- UL/DB2 program environment

Overview

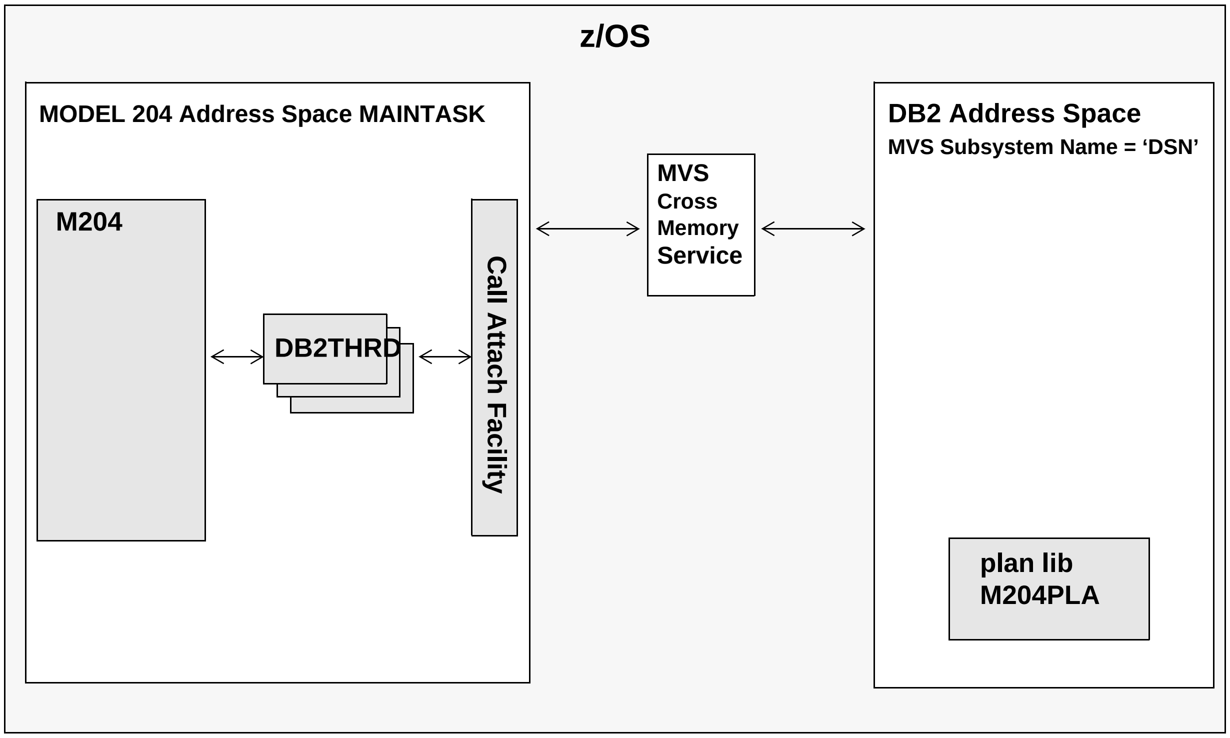
The User Language/DATABASE 2 Interface (UL/DB2) allows a Model 204 User Language procedure to use embedded SQL statements to access a DB2 database. User Language becomes, in effect, a host language for DB2. SQL statements embedded in a User Language procedure can update a DB2 database or return selected values from DB2 to Model 204.

The following example is an SQL statement embedded in a User Language procedure:

```
EXEC DB2
   DECLARE BAR CURSOR FOR
      SELECT CHA FROM DVPJB.TEST2
END EXEC
```

UL/DB2 program environment

Figure 1-1. UL/DB2 program environment



Requirements

The requirements for the User Language/DATABASE 2 Interface are:

- IBM z/OS 1.01 or higher
- Model 204
- DATABASE 2 Version 2 Release 1 or higher
- To use a Model 204 user ID as the DB2 authorization ID, the Model 204 security interface for ACF2 must be installed.

Supported operating systems

UL/DB2 runs under only IBM z/OS. It does not run under IBM z/VM or IBM z/VSE operating systems.

Accessing DB2 address spaces

Model 204 uses the Call Attach Facility (CAF) to communicate with a single local DB2 subsystem that is on the same physical machine as Model 204.

However, this local DB2 subsystem can use DATABASE 2’s Distributed Data Facility (DDF) to communicate with one or more remote DB2 databases. These remote databases can be on the same CPU as the local database or on a different CPU.

Limitations

The following limitations apply to the UL/DB2 Interface:

- UL/DB2 Interface is for only z/OS DB2.
- JCL to run Model 204 must include DSNvrm.DSNLOAD in the STEPLIB in order to load DSNALI and DSNTIAR, unless the module is in the Link Lookaside Area.
- UL/DB2 Interface is for a single machine implementation.
- No support for singleton SELECT statements, although singleton SELECT statements can be issued using the $SPIFY function.
- No system manager commands to control the subtasks.
- Number of threads, or subtasks, is defined statically.
- Model 204 address space can have a link open to only one local DB2 subsystem.
- Maximum number of cursors that you can have open simultaneously is ten.

2

Defining the Model 204 Environment

In this chapter

- Overview
- Preparing a Model 204 environment
- Setting the UL/DB2 User 0 parameters
- Defining the LINK, PROCESSGROUP, and PROCESS
- Controlling links
- Security processing
- Granting PLAN and table privileges
- Using the BUMP command with UL/DB2
- Using the MONITOR command with UL/DB2

Overview

The UL/DB2 Interface is supported in the Online and BATCH204 environments.

An Online is a multiuser program that runs in its own address space. Each Online is a separate entity, requiring its own system files, buffers, control tables, and servers.

BATCH204 is a single-user Model 204 job, therefore, there are no terminal interfaces. Input is only possible from the User 0 input stream. BATCH204 requires its own system files, buffers, control tables, and servers.

Preparing a Model 204 environment

To prepare a Model 204 environment to support UL/DB2, you must:

1. Set the User 0 parameters.
2. Define the LINK, PROCESSGROUP, and PROCESS.
3. Open the link.

These tasks are described in the following sections.

Setting the UL/DB2 User 0 parameters

Model 204 uses User 0 parameters to configure its address space (Online or BATCH204). User 0 parameters are specified in the input stream of the job invoking Model 204.

UL/DB2 has two required and two optional User 0 parameters.

The *required* UL/DB2 User 0 parameters are:

- DB2THRD
- DB2PLAN

You must explicitly set a value for both DB2THRD and DB2PLAN.

The *optional* UL/DB2 User 0 parameters are:

- DB2QUOTE, which has a default of an apostrophe (')
- DB2POINT, which has a default of a decimal point (.)

You can set a value for either or both DB2QUOTE and DB2POINT. If you do not set a value for either DB2QUOTE or DB2POINT, that parameter assumes a default value.

Setting the XMEMOPT and XMEMSVC parameters

If your site uses the ACF2 external security package and you want to take advantage of operating system Cross-Memory Services, you must set two additional User 0 parameters, XMEMOPT and XMEMSVC. See “Security processing” on page 10 for more information about setting these parameters.

Setting the SPCORE parameter

Each UL/DB2 user requires 1300 bytes of SPCORE.

Setting User 0 parameters in the CCAIN input stream

You set the UL/DB2 User 0 parameters on the first line of the CCAIN input stream. The User 0 parameter line is the first line of the CCAIN input data set, after any DEFINE commands, in a Batch or Online system.

The following example shows the positioning of the User 0 parameter line within the CCAIN input stream:

```
//CCAIN DD *
*  User 0 parameters
.
.
.
```

See the *Model 204 System Manager's Guide* and the *Model 204 Command Reference Manual* for more information on setting User 0 parameters.

RESET command not supported

You cannot use the RESET command to reset the UL/DB2 User 0 parameters. You can set them only on the User 0 parameter line in the CCAIN input stream.

VIEW command supported

You can use the VIEW command during an Online run to display the current User 0 parameter settings. See the *Model 204 Command Reference Manual* for more information.

DB2THRD parameter

The DB2THRD parameter specifies the number of DB2 threads (TCBs) to allocate for a Model 204 run.

Syntax `DB2THRD=nnnnn`

Where:

nnnnn is a number that specifies the maximum number of DB2 threads to allocate for a Model 204 run. The number that is best for your site depends on the types of transactions that you process. If you typically perform complex tasks, you might want to allocate a thread for each user. If you typically perform simple tasks, you might want to allocate one thread for every ten users.

Note: If the value of DB2THRD is greater than the value of the NUSERS parameter, Model 204 resets the value of DB2THRD to the value of NUSERS and writes an error message in the job log.

DB2PLAN parameter

The DB2PLAN parameter indicates the PLAN name created by the DBRM BIND.

Syntax `DB2PLAN=`*`name`*

where:

name is the PLAN name created by the DBRM BIND. For more information on the PLAN name, see “Granting PLAN and table privileges” on page 11.

DB2QUOTE parameter

Note: The DB2QUOTE parameter depends on the systemwide definitions made by your DB2 DBA. Consult with your DBA before setting this parameter.

DB2QUOTE is an optional parameter that specifies the symbol to be used as the DB2 string delimiter. DB2QUOTE has a default value of an apostrophe (’).

Syntax `DB2QUOTE=`*`symbol`*

where:

symbol is the character to be used as the DB2 string delimiter.

For more information on the DB2 string delimiter, see the *IBM DATABASE 2 Install Guide.*

DB2POINT parameter

Note: The DB2POINT parameter depends on the systemwide definitions made by your DB2 DBA. Consult with your DBA before setting this parameter.

DB2POINT is an optional parameter that specifies the symbol to be used as the decimal point. DB2POINT has a default value of period (.).

Syntax `DB2POINT=`*`symbol`*

where:

symbol is the character to be used as the DB2 decimal point.

For more information on the DB2 decimal point, see the *IBM DATABASE 2 Install Guide.*

Defining the LINK, PROCESSGROUP, and PROCESS

The following DEFINE commands are required for UL/DB2. The DEFINE commands identify the entities that underlie UL/DB2.

These commands are fully documented in the *Model 204 Command Reference Manual*:

- DEFINE LINK: UL/DB2
- DEFINE PROCESSGROUP: UL/DB2
- DEFINE PROCESS: User Language to DATABASE 2

Guidelines for using the DEFINE commands

The following guidelines apply when using the DEFINE commands.

- User zero or system manager privileges are required
- Typically, you place the DEFINE commands in the User 0 stream in the CCAIN input file. This placement allows the entities to be defined when Model 204 starts.

  Alternatively, you can place the DEFINE commands inside a Model 204 procedure that is invoked within the User 0 stream. This allows you to set up or change the definitions without restarting Model 204.
- You can specify the DEFINE commands in any order.

Controlling links

The following commands that you use to control links between Model 204 and DB2 are fully documented in the *Model 204 Command Reference Manual:*

- The OPEN LINK command enables the link specified in the DEFINE LINK command. It is required to establish the connection between Model 204 and DB2.
- The CLOSE LINK command disables an open link to prohibit users from accessing DB2 after initialization has completed.
- The STOP LINK command sets the link entity in a drain state. That is, no new users are allowed to access DB2, and current users can continue until they end the connection normally.

Placing the LINK commands

You can place these commands in the User 0 stream, within a User Language procedure, or you can issue them at the command level.

Required privileges

To issue these commands, you must have User 0, system manager, or system administrator privileges.

Security processing

To use DB2 resources, a Model 204 user must be authorized to access the DB2 subsystem and, once that connection has been established, to access specific DB2 resources. This section describes how DB2 determines if a Model 204 user has access authorization, both with and without an external security package installed.

To access the DB2 subsystem, Model 204 supplies a primary authorization ID to DB2 at connection time. How this processing is done depends on whether an external security package is in place.

No external security in place

If there is no external security system, DB2 uses the USER parameter on the JOB statement as the primary authorization identifier. If there is no USER parameter, the primary authorization ID is set to the default authorization ID, which was set when DB2 was installed (UNKNOWN AUTHID on install panel DSNTIPP).

External security in place

An external security system can be used to validate DB2 requests.

To fully use an external security subsystem to validate the DB2 requests, one of the Model 204 security interfaces must be installed and the Model 204 load member must be in an APF-authorized load library. If the load library is not authorized, then DB2 validates the request as if there were no external security system in place. See the previous section, “No external security in place”.

Taking advantage of operating system Cross-Memory Services

If you have an external security subsystem installed and are running Model 204 from an authorized library, then you can take advantage of operating system Cross-Memory Services. To do so, you must:

- Have Cross-Memory SVC installed. See the *Model 204 z/OS Installation Guide* for information on installing XMEMSVC.
- Set two User 0 parameters to ensure that a Model 204 run uses cross-memory. These User 0 parameters are XMEMOPT and XMEMSVC.
  - Set XMEMOPT to `X'04'` to force Model 204 to initialize the Cross-Memory environment.
  - Set XMEMSVC to the SVC number used when XMEMSVC was installed.

  Note: If XMEMOPT has already been set for another Model 204 feature, then you do not need to set it again.

Authorization exit DSN3@ATH

The authorization exit DSN3@ATH is provided by DB2 for security checking during CONNECT processing. The exit as distributed checks for security subsystem installation, and uses the security subsystem to validate the primary authorization ID and acquire any secondary authorization ID. CA-ACF2 replaces this exit with one of their own, which does identical work, albeit by calling on ACF2 functions.

As part of UL/DB2, Rocket Software distributes its own version of the DSN3@ATH authorization exit. You must install this Model 204-supplied authorization exit, which is stored in the MACLIB as ACF3SATH, over the existing DSN3@ATH. Refer to Chapter 3 of the *DATABASE 2 Administration Guide Volume 1* for instructions on installing the Model 204-distributed DSN3@ATH over the existing copy.

Note: If you have modified the existing DSN3@ATH, you must make the same modifications to the Model 204 DSN3@ATH authorization exit.

When this exit is invoked, the active TCB is checked to see if it is a Model 204 subtask. If this TCB is recognizable as Model 204's, the Model 204 user ID is used as the DB2 primary authorization ID (AIDLPRIM) and the default SQL ID.

The authorization exit is driven at connection time, that is, during the Call Attach Facility CONNECT call. CONNECT processing is expensive, so user switching requires careful consideration. A user procedure acquires a subtask TCB at the first EXEC DB2 request and keeps it until the procedure finishes. A long-running procedure can effectively remove the TCB from the pool of available TCBs.

Granting PLAN and table privileges

To execute a program containing SQL statements, a Model 204 user must have execution privileges on the PLAN that was created by the BIND. The DB2 DBA can grant privileges to individual user IDs, can make the PLAN public, or do whatever is standard at the site. Also, to access specific DB2 tables, the Model 204 user must have appropriate table privilege for that table. Again, the DB2 DBA grants table privileges to users.

Using the BUMP command with UL/DB2

You can use the Model 204 BUMP command to terminate a UL/DB2 user.

Depending on the state of that user, however, the BUMP command might have to wait to complete processing. A procedure that is accessing DB2 runs under the subtask's TCB. Model 204 considers that user to be in a swappable WAIT state. When you issue a BUMP against such a user, UL/DB2 waits until the user is swapped in and active before executing the BUMP. The user is then restarted, which frees the subtask TCB.

See the *Model 204 Command Reference Manual* for more information on the BUMP command.

Using the MONITOR command with UL/DB2

Use the MONITOR command to see the current usage of a Model 204 link, processgroup, or process.

Syntax

```
MONITOR {LINK | PROCESSGROUP | PROCESS} entityname
        [[EVERY] n]
```

where:

entityname is the name specified on the DEFINE LINK, DEFINE PROCESSGROUP, or DEFINE PROCESS command.

n is the number of seconds Model 204 waits after completing a display before beginning the next display.

Output The output of the MONITOR command for UL/DB2 is a single line that contains the items listed in Table 2-1.

Table 2-1. MONITOR command output

Term	Meaning
LOCAL ID	Blank (no meaning)
MAXSES	Maximum number of threads allowed. The value of the DB2THRD parameter.
BNDSES	Number of threads currently in use. (Threads that are in use or on the chain.)
CONVS	Threads currently active to DB2. (WAIT type 32. Model 204 WAIT, the thread is talking to DB2.)
FLGS	Link status flags: A — Active S — Stopped C — Being closed
TRAN	Type of transport: LOCL — Local
PROTO	Type of communication protocol: CAF — Call Attach Facility

See the *Model 204 Command Reference Manual* for more information about the MONITOR command.

3

Coding SQL Statements in a User Language Procedure

In this chapter

Overview

This chapter describes the specific SQL statements that you can use in a User Language procedure, and the rules and guidelines that apply when using those statements.

The chapter also describes the $SPIFY interface, which lets a User Language programmer test an SQL statement before it is used in a procedure.

Prerequisite knowledge of SQL

This chapter assumes that you are familiar with SQL syntax. It does not explain what a particular SQL statement does; rather, it explains how you use that statement in a User Language procedure.

Requirements for writing SQL statements

This section describes the general requirements that apply when you code an SQL statement in a User Language procedure.

Components of an SQL statement in a User Language procedure

An SQL statement can contain constants, SQL column names, and Model 204 %variables. An SQL statement cannot contain image names, screen names, or Model 204 field names.

EXEC DB2... END EXEC delimiters

You must precede each SQL statement in a User Language procedure with the keywords “EXEC DB2”, and you must end each SQL statement with the keywords “END EXEC”.

You can put the EXEC DB2 and END EXEC on a separate line from the rest of the statement, or you can include it on the same line. You cannot, however, break either the EXEC DB2 or the END EXEC strings onto more than one line.

Two valid SQL statements are shown below (the indentation has been added for legibility):

```
EXEC DB2
   COMMIT WORK
END EXEC


EXEC DB2 FETCH P INTO %POLN END EXEC
```

Continuing SQL statements

You do not need to use a continuation character to continue an SQL statement onto the following line.

Examples

```
EXEC DB2
DECLARE FOO CURSOR FOR SELECT
S#,SNAME,STATUS,CITY FROM S
END EXEC
```

You can continue a literal by placing a hyphen at the end of the line:

```
EXEC DB2
```

```
   DELETE FROM DVPJB.S
   WHERE NAME = 'ALFRED E. -
      NEWMAN'
END EXEC
```

Using quotation marks

Use single quotation marks (’) around literal text in an SQL statement.

Example

```
EXEC DB2
   DELETE FROM DVPJB.S
   WHERE NAME = 'ALFRED E. -
     NEWMAN'
END EXEC
```

Note: This example assumes the default value of the DB2QUOTE parameter, which is an apostrophe (’). If your site has changed the value of DB2QUOTE, use that value instead.

SQL comments not supported

The UL/DB2 Interface does not support SQL comments.

Using multiple SQL statements in a procedure

You can have any number of SQL statements in a single User Language procedure. However, you can code only one SQL statement in each EXEC DB2... END EXEC block.

Example

```
EXEC DB2
   SQL statement
END EXEC
...
EXEC DB2
   SQL statement
   statement continued
END EXEC
...
EXEC DB2
   SQL statement
END EXEC
```

Checking $STATUS

Check $STATUS after each EXEC DB2... END EXEC block to make sure that the processing completed correctly. See “$STATUS and $STATUSD” on page 32 for more information on checking $STATUS in a User Language procedure.

Connecting a User Language procedure to DB2

This section describes the CONNECT TO and DISCONNECT FROM statements that you use to create and terminate a connection, or thread, between a User Language procedure and DB2.

Strictly speaking, CONNECT TO and DISCONNECT FROM are not SQL statements, but Call Attach Facility calls. However, the same rules apply to using these statements in a User Language procedure that were described in the section “Requirements for writing SQL statements” on page 14.

You issue the CONNECT TO and DISCONNECT FROM statements relative to the way that you have defined the process for your UL/DB2 environment. See the DEFINE PROCESS command for User Language to DATABASE 2 in the Model 204 Command Reference Manual.

CONNECT TO statement

Use the CONNECT TO statement to create a thread between a User Language procedure and DB2.

CONNECT TO must be the first EXEC DB2 statement in the procedure. If CONNECT TO is not the first EXEC DB2 statement in a procedure, the results are unpredictable.

Syntax

```
EXEC DB2
   CONNECT TO symbolic_name
END EXEC
```

where:

symbolic_name is the symbolic name from the DESTINATION parameter of the DEFINE PROCESS command.

Errors connecting to DB2

If the CONNECT TO statement within a User Language procedure fails, Model 204 displays an error message. See “Errors involving Call Attach Facility calls” on page 32 for more information.

DISCONNECT FROM statement

The DISCONNECT FROM statement is optional, but is highly recommended if the procedure is long running and you do not require any more access to DB2. The DISCONNECT statement releases system resources for other users.

Syntax

```
EXEC DB2
   DISCONNECT FROM symbolic_name
END EXEC
```

where:

symbolic_name is the symbolic name from the DESTINATION parameter of the DEFINE PROCESS command.

A DISCONNECT is performed for the user at user logout time, or when the subtask control block is “stolen” for use by another DB2 request. Issue the DISCONNECT command only when no more DB2 requests are required by the running procedure.

See Appendix A for more information.

Defining and using cursors

This section describes the SQL statements that you can use to define a cursor and to manipulate the row at which the cursor points. This section also discusses multiple cursor support.

Supported cursor statements

The SQL cursor operation statements supported by UL/DB2 are:

- DECLARE
- OPEN
- FETCH
- CLOSE

In addition, two other supported statements use cursors:

- UPDATE ... CURRENT (positioned UPDATE)
- DELETE ... CURRENT (positioned DELETE)

These statements are described in “Modifying DB2 tables” on page 20.

Managing cursors

You must place statements that declare a cursor before any statement that references that cursor.

Try to close cursors in your application as soon as they are no longer needed. Closing a cursor minimizes the system resources required by your application.

DECLARE statement

Use the DECLARE statement to declare a cursor and its associated query.

Syntax

```
EXEC DB2
   DECLARE cursor_name CURSOR
   FOR SELECT query_expression
END EXEC
```

where:

cursor_name is from 1-18 characters.

See “Using %variables in SQL statements” on page 24 for more information about the use of %variables in a query_expression.

Example

```
EXEC DB2
   DECLARE FOO CURSOR FOR
   SELECT P#,PNAME,CITY
   FROM DVPJB.P
END EXEC
```

This block of code associates a cursor “FOO” named with the results that are returned by the SELECT statement “SELECT P#, PNAME, CITY FROM DVPJB.P”.

OPEN statement

Use the OPEN statement to prepare a cursor for processing.

Syntax

```
EXEC DB2
   OPEN cursor_name
END EXEC
```

where:

cursor_name is a cursor that has been declared earlier in the procedure. When the OPEN statement is executed, the values of any %variables are substituted in the cursor declaration.

Example

```
EXEC DB2
   OPEN FOO
END EXEC
```

This block opens the cursor FOO, declared in the previous example.

Multiple cursor support

You can open and use more than one cursor simultaneously. The maximum number of cursors that you can have open simultaneously is 10.

A request for more than 10 cursors results in a compile-time error.

FETCH statement

Use the FETCH statement to place a single row of returned values into a %variable.

Syntax

```
EXEC DB2
   FETCH cursor_name INTO
      %variable {{{INDICATOR} %variable},…}
END EXEC
```

where:

- *cursor_name* is the name of a cursor previously declared and opened.
- *INDICATOR keyword is optional when the Indicator %variable is present*
- *%variable* is a User Language %variable that has been previously declared. The number of %variables must match number of SQL columns named in the query.

  See “Using %variables in SQL statements” on page 24 for more information.

Example The following block of code fetches P#, PNAME, and CITY from the table PVBJB.P and assigns them to the User Language variables %A, %B, and %C:

```
REPEAT FOREVER
   EXEC DB2
      FETCH FOO INTO %A,%B,%C
   END EXEC
   IF $STATUS NE 0 AND $STATUS NE 100 THEN
      JUMP TO ERROR
   END IF
   PRINT ’NUMBER:’ AND %A AND ’ NAME:’ AND %B AND
   ’ CITY:’ AND %C
END REPEAT
```

A $STATUS of 100 indicates that the FETCH found no rows or no more rows. After each FETCH, test for a $STATUS of 100 and move out of the loop when you encounter it. See Chapter 4 for more information on $STATUS and error processing.

Also, test for $STATUS NE 0 and $STATUS NE 100 for a real error condition (an exceptional case).

CLOSE statement

The CLOSE statement closes a cursor that has been previously opened. To reduce the overhead of your program, close a cursor as soon as it is no longer required for your program.

Syntax

```
EXEC DB2
   CLOSE cursor_name
END EXEC
```

Example The following example closes the cursor FOO:

```
EXEC DB2
   CLOSE FOO
END EXEC
```

Modifying DB2 tables

This section describes the SQL statements that you can use in a User Language procedure to insert, update, or delete data in a DB2 table.

The supported SQL statements that modify a DB2 table are:

- INSERT
- UPDATE (searched UPDATE)
- DELETE (searched DELETE)
- UPDATE ... CURRENT (positioned UPDATE)
- DELETE ... CURRENT (positioned DELETE)

INSERT statement

Use the INSERT statement to insert rows into a table one at a time, or to copy data from one table to another, processing multiple rows with a single statement.

Inserting Large Object fields

In this release the INSERT statement is not supported for Large Object fields.

Inserting a single row

Use the following syntax of the INSERT statement to processes a single row at a time.

Syntax

```
EXEC DB2
   INSERT INTO table_name VALUES ([%variable|lit-
eral|NULL],...)
END EXEC
```

where:

%*variable* is a User Language %variable.

Example The following code inserts the values “LONDON”, “BUDDHA”, “S9”, and “3” into the table DVPJB.S:

```
DECLARE %ERR STRING LEN 240
DECLARE %NAME STRING LEN 20
DECLARE %CITY STRING LEN 10
DECLARE %STAT FIXED
DECLARE %SER STRING LEN 5
%CITY = 'LONDON'
%NAME = 'BUDDHA'
%SER  = 'S9'
%STAT = 3
EXEC DB2
   CONNECT TO BOSTON
END EXEC
IF $STATUS NE 0 THEN
   JUMP TO CONERROR
END IF
EXEC DB2
   INSERT INTO DVPJB.S
   VALUES (%SER)
END EXEC
IF $STATUS NE 0 THEN
   JUMP TO ERROR
END IF
```

Inserting multiple rows

The following syntax of the INSERT statement that processes multiple rows.

Syntax

```
EXEC DB2
   INSERT INTO table_name column_list
   SELECT query_specification
END EXEC
```

See "Using %variables in SQL statements" on page 24 for more information about the use of %variables in a query_specification.

You can use this construct to copy data from one DB2 table to another.

Example The following code selects the SNAME, STATUS, and CITY fields from the S table and copies them into the table DVPJB.S:

```
EXEC DB2
   CONNECT TO BOSTON
END EXEC
IF $STATUS NE 0 THEN
   JUMP TO CONERROR
END IF
EXEC DB2
   INSERT INTO DVPJB.S (SNAME, STATUS, CITY)
   SELECT SNAME, STATUS, CITY FROM S
END EXEC
IF $STATUS NE 0 THEN
   JUMP TO ERROR
```

```
END IF
```

Searched UPDATE statement

Use the searched UPDATE statement to update one row of a table at a time.

Syntax

```
EXEC DB2
   UPDATE table_name
   SET {column = [%variable|literal|NULL]},...
   WHERE search_condition
END EXEC
```

where:

%variable is a previously declared User Language %variable.

See “Using %variables in SQL statements” on page 24 for more information about the use of %variables in a search_condition.

Example

```
PRINT 'UPDATE'
EXEC DB2
   UPDATE S SET SNAME = 'BILL', CITY = 'SAN DIEGO' WHERE
S# = %S
END EXEC
IF $STATUS NE 0 THEN
   JUMP TO ERROR
END IF
```

Searched DELETE statement

Use the searched DELETE statement to delete one row or a table at a time.

Syntax

```
EXEC DB2
   DELETE FROM table_name
   WHERE column = search_condition
END EXEC
```

where:

%variable is a previously declared User Language %variable.

See “Using %variables in SQL statements” on page 24 for more information about the use of %variables in a search_condition.

Example

```
DECLARE %ERR STRING LEN 240
DECLARE %NAME STRING LEN 20
DECLARE %CITY STRING LEN 10
DECLARE %STAT FIXED
DECLARE %SER  STRING LEN 5
%CITY = 'LONDON'
```

```
%NAME = 'BUDDHA'
%SER  = 'S9'
%STAT = 3
EXEC DB2
   DELETE FROM DVPJB.S
   WHERE CITY = %CITY
END EXEC
IF $STATUS NE 0 THEN
   JUMP TO ERROR
END IF
```

UPDATE... CURRENT statement

Use the UPDATE... CURRENT, as follows.

Syntax

```
EXEC DB2
   UPDATE table_name
   SET {column = [%variable|literal|NULL]},...
   WHERE CURRENT OF cursor_name
END EXEC
```

where:

%variable is a previously declared User Language %variable.

Example

```
EXEC DB2
   DECLARE FOO CURSOR FOR
   SELECT S#,SNAME,CITY FROM DVPJB.S
   WHERE CITY = %CITY
   FOR UPDATE of SNAME
END EXEC
*
EXEC DB2
   OPEN FOO
END EXEC
IF $STATUS NE 0 THEN
   JUMP TO ERROR
END IF
*
REPEAT FOREVER
   EXEC DB2
      FETCH FOO INTO %A,%B,%C
   END EXEC
    IF $STATUS = 100 THEN LOOP END
    END IF
   IF $STATUS NE 0 THEN
      JUMP TO ERROR
   END IF
   PRINT 'NUMBER:' AND %A AND ' NAME:' AND %B AND '
   CITY:’ AND %C
   EXEC DB2
      UPDATE DVPJB.S
      SET SNAME = %NAME
      WHERE CURRENT OF FOO
   END EXEC
   IF $STATUS NE 0 THEN
      JUMP TO ERROR
   END IF
END REPEAT
```

DELETE... CURRENT statement

Use the DELETE... CURRENT statement as follows.

Syntax

```
EXEC DB2
   DELETE FROM table_name
   WHERE CURRENT OF cursor_name
END EXEC
```

Example

```
REPEAT FOREVER
   EXEC DB2
      FETCH FOO INTO %A,%B,%C
   END EXEC
   IF $STATUS = 100 THEN LOOP END
   END IF
   IF $STATUS NE 0 THEN
      JUMP TO ERROR
   END IF
   PRINT ’NUMBER:’ AND %A AND ’ NAME:’ AND %B AND ’ CITY:’
   AND %C
   IF $DEBLANK(%B,1) EQ %NAME THEN
      PRINT ’DELETING’ AND %B
   EXEC DB2
      DELETE FROM DVPJB.S
      WHERE CURRENT OF FOO
   END EXEC
   IF $STATUS NE 0 THEN
      JUMP TO ERROR
   END IF
   END IF
END REPEAT
```

Using %variables in SQL statements

This section explains how you use User Language %variables in SQL statements. It also discusses how to specify the INDICATOR type with the FETCH command.

%variables

You can use a User Language %variable in an SQL statement to provide a value to DB2, or to receive the result of an SQL's statement processing.

Do not place a colon before a %variable.

Using %variables for Input

The SQL statements that use a %variable to provide a value to DB2 are:

- UPDATE... SET.. CURRENT
- UPDATE... SET ... WHERE
- INSERT... VALUES
- DELETE ... WHERE
- DECLARE .. CURSOR

UL/DB2 parses the input %variables to determine type and value. You can use a numeric %variable wherever a number can appear in an SQL statement. You can use a string %variable wherever a quoted string can appear in an SQL statement.

Example The %CITY variable adds the value “LONDON” to the DVPJB.S table:

```
DECLARE %CITY STRING LEN 10
%CITY = 'LONDON'
EXEC DB2
   CONNECT TO BOSTON
END EXEC
IF $STATUS NE 0 THEN
   JUMP TO CONERROR
END IF
EXEC DB2
   INSERT INTO DVPJB.S (CITY)
   VALUES (%CITY)
END EXEC
```

Using %variables for output (FETCH statement)

The SQL FETCH statement uses a %variable to receive output.

The output is assigned to the %variables in the order in which it is returned from DB2. If data is truncated when assigned to the %variable, $STATUS and $STATUSD indicate that truncation occurred. If more columns are available from the query row than there are %variables specified, the output data is still put in the %variables in the order returned, but the $STATUS and $STATUSD indicate that more data was available.

Example %A, %B, and %C in the following code receive the values of the P#, PNAME, and CITY columns from the table PVBJB.P:

```
REPEAT FOREVER
   EXEC DB2
      FETCH FOO INTO %A,%B,%C
   END EXEC
   IF $STATUS NE 0 AND $STATUS NE 100 THEN
      JUMP TO ERROR
   END IF
   PRINT 'NUMBER:' AND %A AND ' NAME:' AND %B AND ' CITY:'
   AND %C
END REPEAT
```

Stripping blanks from string values returned by DB2

When DB2 returns a string to a User Language %variable, that string contains the number of blanks necessary to make the value equal to the defined maximum length of the %variable.

Example In the following code, if the string being fetched by CUR1 into %BAR is three characters ("Joe"), that value is padded with enough blanks to make it 20 characters long ("Joe   "):

```
DECLARE %BAR STRING LEN 20
FETCH CUR1 INTO %BAR
$LEN(%BAR) = 20
```

A User Language string with a length of 20, that is assigned a 3-character value ("Joe") has a length of three:

```
DECLARE %FOO STRING LEN 20
%FOO =' JOE'
$LEN (%FOO) = 3
```

As a result, an equality comparison between the values of %FOO and %BAR is false. If you want a string value returned by DB2 to conform to the way that User Language construes string variables, you must use the $DEBLANK function to strip the extraneous blanks:

```
$DEBLANK(%BAR)
```

Data types

The User Language/DATABASE 2 Interface supports string, float, and fixed data types; these are all Model 204 data types. The assignment to and from these variable types follows the rules outlined in the *IBM DATABASE 2 SQL Reference.*

Specifying a numeric type when a string type is expected, or specifying a string type when a numeric type is expected, generates an error. The error code is returned by $STATUS and $STATUSD.

Specifying the INDICATOR type with the FETCH command

You can specify the INDICATOR type on FETCH commands.

Syntax

```
EXEC DB2
   FETCH cursor_name
   INTO %variablea INDICATOR %indica, %variableb
      INDICATOR %indicb,...
END EXEC
```

where:

%indica in this statement is the indicator variable for *%variablea*.

INDICATOR is an optional keyword; if a %variable is followed directly by a %variable with no intervening comma, it is assumed to be an indicator variable.

Example

```
EXEC DB2
FETCH cursor_name
INTO %variablea %indica, %variableb
END EXEC
```

Here, *%indica* is the indicator variable for *%variablea*.

Querying the INDICATOR variable

In UL/DB2, the User Language procedure must query the INDICATOR variable to see if the NULL indicator was set (-1). If the column is NULL and no indicator variable is present, $STATUSD is set to -305 (the SQLCODE). Retrieving $DB2EMSG displays the text associated with SQLCODE -305. The receiving %variable is not set when a null value is fetched.

Terminating transactions

UL/DB2 supports the COMMIT and ROLLBACK statements to terminate transactions.

COMMIT statement

A commit point occurs when you issue the COMMIT statement, or when the request ends normally.

Syntax

```
EXEC DB2
   COMMIT WORK
END EXEC
```

The exception to this rule is the APSY AUTOCOMMIT=NO condition. In this case, there is no implicit COMMIT at procedure end; a DB2 COMMIT is issued only if a Model 204 User Language COMMIT statement is issued.

ROLLBACK statement

UL/DB2 issues a ROLLBACK statement if either Model 204 or DB2 terminates abnormally.

Syntax

```
EXEC DB2
   ROLLBACK WORK
END EXEC
```

$SPIFY interface

$SPIFY is a Model 204 function that you can use in a User Language request to issue SQL commands. $SPIFY is designed to give the User Language programmer an environment similar to the one that SQL Processor Using File Input (SPUFI) provides to DB2 application programmers in the TSO environment. $SPIFY gives you, in effect, a DB2 command line.

Using $SPIFY, you can issue Data Definition commands to set up tables and views, and process SELECT statements. You can include multiple $SPIFY functions in a procedure.

$SPIFY takes any valid SQL statement as an argument; the string must follow Model 204 string conventions. The argument can be either a quoted string or a string %variable.

Syntax

```
PRINT $SPIFY(DB2-command-string)
```

Example

```
EXEC DB2
CONNECT TO BOSTON
END EXEC
PRINT $SPIFY('SELECT * FROM DVPJB.S')
```

Output The result of the $SPIFY function is a series of print lines. The last line is printed without an end-of-line; therefore, the recommended usage is PRINT $SPIFY(*DB2-command-string*). The output produced depends on whether the *DB2-command-string* is a valid SELECT statement.

Valid SELECT statement

The output produced for a valid SELECT statement is:

- Printed copy of the argument, that is, the SELECT statement
- Column headers
- Selected rows

- Number of rows printed
- Line indicating successful execution and the final SQLCODE (100, indicating the end of the selected rows)

Valid statement other than SELECT

The output produced for a valid statement other than SELECT is:

- Printed copy of the argument, that is, the DB2 request
- Line indicating successful execution and the final SQLCODE (0)
- Line indicating that the call has finished

Invalid statement

The output produced for an invalid statement is:

- Printed copy of the argument, that is, the invalid statement, preceded by ’SQL REQUEST:’
- Several lines containing the DB2 error message

Note: In $SPIFY, a dash (-) in the output indicates that the column has the NULL indicator set.

You can use the Model 204 USE PRINTER command to direct the output. See the *Model 204 Command Reference Manual* for more information about the USE PRINTER command.

4

Interpreting Codes and Messages

In this chapter

- Overview
- Errors involving Call Attach Facility calls
- Errors involving SQL statements

Overview

This chapter describes how you interpret the return codes and messages returned by the User Language/DATABASE 2 Interface.

UL/DB2 error processing

When a UL/DB2 statement executes, UL/DB2 sets status information that indicates whether the statement completed normally. UL/DB2 communicates this status information through the User Language $STATUS and $STATUSD functions. UL/DB2 communicates the error message associated with an abnormal status code through the $ERRMSG function for Call Attach Facility (CAF) calls, and the $DB2EMSG function for SQL statements.

Using tracing

UL/DB2 error processing is handled by CAF. If you allocate a DSNTRACE data set for the job, CAF sends diagnostic messages to the DSNTRACE data set.

Typically, you do not want to turn DSNTRACE on. If you turn DSNTRACE on and have multiple users, you might get a S013E ABEND. This is an IBM problem that is caused by DSNTRACE not being able to work with multiple tasks. There is an APAR — a suggestion for IBM to fix the problem.

Turn DSNTRACE on only when debugging in a single user environment. To turn DSNTRACE on, add a DD card to the startup JCL.

See the *IBM DATABASE 2 Call Attach Facility User's Guide and Reference* for more information.

Errors involving Call Attach Facility calls

This section describes the diagnostic information returned by UL/DB2 when there is an error that involves a CAF call. The CAF calls that you can use in a User Language procedure are:

- CONNECT TO
- DISCONNECT FROM

$STATUS and $STATUSD

When there is an error involving a CAF call, CAF places a return code in R15, and a reason code in R0. $STATUS retrieves the return code, the contents of R15, after each CAF call. A value of one ($STATUS=1) or greater indicates that an error occurred and identifies the error as belonging to a particular category.

If the return code does not equal zero ($STATUS NE 0), then $STATUSD retrieves the reason code (the contents of R0). $STATUSD indicates the specific error condition within a particular $STATUS category.

Table 4-1 on page 33 lists some of the more common return and reason codes. You can find a complete list of reason and return codes in the *IBM DATABASE 2 Call Attach Facility User's Guide and Reference,* and the *IBM DATABASE 2 Messages and Codes Manual.*

If the return code is equal to zero ($STATUS = 0), $STATUSD returns unpredictable codes.

$ERRMSG

Use the User Language $ERRMSG function to retrieve the error message for an error with a CAF call, that is, CONNECT TO or DISCONNECT FROM. The string returned by $ERRMSG can be up to 80 characters long.

Table 4-1. Common return and reason codes

$STATUS	$STATUSD		Description
	Hexadecimal	Decimal	
0	0	0	Successful completion
4	00C10823	12650531	Release level mismatch
4	00C10824	12650532	CAB Reset complete
4	00F30025	15925285	Subsystem is stopping
8	00000014	20	No MODEL 204 link OPENed
8	00000015	21	MODEL 204 link STOPped
8	00000016	22	MODEL 204 link CLOSEd
8	0000001F	31	CONNECT must be the first DB2 command
8	00000020	32	Second CONNECT from the same procedure
8	00000028	40	No available virtual memory
8	00000032	50	No threads available
8	0000005A	90	DB2 Terminate ECB was posted
8	00F30002	15925250	DB2 subsystem is not available
8	00F30013	15925267	User not authorized to DB2
8	00F30040	15925312	Resource allocation error: unavailable
8	00F30049	15925321	TCB already connected
8	00F30055	15925333	Max connections reached
12	00F30006	15925254	DB2 subsystem name invalid
12	00F30040	15925312	Resource allocation error: invalid
200	00C10201	12648961	Second CONNECT from one TCB
200	00C10202	12648962	Second OPEN from one TCB
200	00C10203	12648963	CLOSE issued with no active OPEN
200	00C10204	12648964	DISCONNECT issued with no active OPEN
200	00C10205	12648965	TRANSLATE issued with no connection
200	00C10206	12648966	Wrong number of parameters
200	00C10207	12648967	Unrecognized action parameter
200	00C10208	12648968	Request to talk to two SSIDs from one TCB
256	00F30018	15925272	TCB not connected to DB2

Example The following code fragment shows how you can test the value of $STATUS after a CAF call:

```
EXEC DB2
   CONNECT TO BOSTON
END EXEC
IF $STATUS NE 0 THEN
   PRINT ’ $STATUS=’ AND $STATUS
   PRINT ’$STATUSD=’ AND $STATUSD
   PRINT $ERRMSG
JUMP TO FINI:
END IF
```

Errors involving SQL statements

This section describes the diagnostic information returned by UL/DB2 when there is an error involving an SQL statement.

Using the $STATUS and $STATUSD functions

After invoking an SQL statement, $STATUS indicates whether an error occurred processing the statement. If no errors occurred, $STATUS is equal to zero. If an error occurred, $STATUS is not equal to zero and the value of STATUSD contains more specific information about the error.

When DB2 executes an SQL statement, it places the SQL return code in the SQLCODE field of the SQL Communication Area (SQLCA). $STATUSD retrieves the value of the SQLCODE field after an SQL statement. Typically, a negative SQL return code indicates an error, a positive return code indicates an exceptional but valid condition, and zero indicates successful execution.

Note: In one special case, $STATUS has the same value as SQLCODE. A $STATUS of 100 indicates table empty or completed; in this case, $STATUSD, which contains SQLCODE, is also 100.

Understanding the $DB2EMSG function

If the SQL return code does not equal zero (SQLCODE NE 0), then DB2 also returns tokens in the SQLCA. UL/DB2 processes the tokens internally, and returns them via $DB2EMSG. The string returned by $DB2EMSG can be up to 240 characters long.

For example, attempting to access a column that does not exist returns an SQLCA with the SQLCODE set to -205, and SQLERRM contains two tokens: the column name and the table name. Looking up the SQLCODE in the *DATABASE 2 Messages and Codes Manual* gives the following error message:

```
-205column-name IS NOT A COLUMN OF TABLE table-name
```

Retrieving the value of $DB2EMSG returns:

```
DSNT408I SQLCODE = -205, ERROR: FOO IS NOT A COLUMN OF
TABLE M204.BAR
```

(This example assumes that DB2 contains table M204.BAR, which does not contain column FOO.)

You can find all the SQL return codes in the *IBM DATABASE 2 Messages and Codes Manual.*

Example The following code fragment shows how you can test the value of $STATUS after executing an SQL statement:

```
EXEC DB2
   OPEN BAR
END EXEC
IF $STATUS NE 0 THEN
   PRINT ’ $STATUS=’ AND $STATUS
   PRINT ’$STATUSD=’ AND $STATUSD
   PRINT $DB2EMSG
END IF
```

5

Sample UL/DB2 Request

In this chapter

- Overview
- Sample request described
- Sample request code
- Sample request output

Overview

This chapter provides a complete UL/DB2 request that connects to a DB2 subsystem and uses the $SPIFY function to access a DB2 table. The sample request illustrates much of the material discussed in the earlier chapters. Specifically, the sample request demonstrates using:

- Model 204 DEFINE and OPEN LINK commands
- CAF CONNECT TO call
- User Language $STATUS, $STATUSD, and $ERRMSG functions
- UL/DB2 $SPIFY function

Sample request described

The example contains two procedures: DB2PRC and SPIFY. The function of each procedure is described in the following sections.

DB2PRC procedure

The DB2PRC procedure defines the required UL/DB2 entities and enables the link to DB2. Specifically, the DB2PRC procedure:

- Uses the DEFINE commands to establish the link, processgroup, and process
- Uses the OPEN LINK command to enable the link between the User Language procedure and the DB2 subsystem

In the following example, note that the link name specified in the DEFINE LINK, DEFINE PROCESSGROUP, and OPEN LINK commands must be the same (in this example, DB2LNK).

Also, note that the processgroup name specified in the DEFINE PROCESSGROUP and DEFINE PROCESS commands must be the same (in this example, DB2GRP).

SPIFY procedure

The SPIFY procedure connects the User Language request to a DB2 subsystem and retrieves data from a DB2 table. Specifically, the SPIFY procedure:

- Uses the CONNECT TO call to create a thread between a User Language procedure and DB2.

  The value specified in the CONNECT TO call must be the same value given in the DESTINATION parameter of the DEFINE PROCESS command (in this example, BOSTON).

  Also, the CONNECT TO call must be the first EXEC DB2 statement in the procedure.
- Uses the $STATUS function to determine if the CONNECT was successful. If it failed, the procedure returns diagnostic information through the $STATUS, $STATUSD, and $ERRMSG functions.
- Uses the $SPIFY function to select data from a DB2 table.

Sample request code

Note: Before running this procedure, you must set the required UL/DB2 User 0 parameters, DB2THRD and DB2PLAN, in the CCAIN input stream. See “Setting the UL/DB2 User 0 parameters” on page 6 for more information on User 0 parameters.

The code for the sample request is:

```
PROCEDURE DB2PRC
DEFINE LINK DB2LNK WITH SCOPE=SYSTEM TRANSPORT=INTERNAL -
       PROTOCOL=CAF
DEFINE PROCESSGROUP DB2GRP WITH SCOPE=SYSTEM LINK=DB2LNK -
       DB2ID=DSN
DEFINE PROCESS CCADB2 WITH SCOPE=SYSTEM -
       DESTINATION=(DB2GRP,BOSTON)
OPEN LINK DB2LNK
END PROCEDURE

PROCEDURE SPIFY
B
*
EXEC DB2
   CONNECT TO BOSTON
END EXEC
IF $STATUS NE 0 THEN
   PRINT ' $STATUS=' AND $STATUS
   PRINT '$STATUSD=' AND $STATUSD
   PRINT $ERRMSG
   JUMP TO FINI
END IF
*
PRINT $SPIFY('SELECT * FROM ??TABLE')
*
FINI:
END
END PROCEDURE
```

The SPIFY procedure uses the User Language ?? construct. When the request is run, the user is prompted to supply a value for the table name.

The output from this request is shown in the section “Sample request output” on page 40.

Sample request output

The output from the sample request is shown in Figure 5-1. A value of “TDEPT” was supplied for the table name.

Figure 5-1. Sample request output

```
SELECT * FROM TDEPT
---------+---------+---------+---------+---------+---------+----
DEPTNO  DEPTNAME                             MGRNO   ADMRDEPT
---------+---------+---------+---------+---------+---------+----
A00     SPIFFY COMPUTER SERVICE DIV.         000010  A00
B01     PLANNING                             000020  A00
C01     INFORMATION CENTER                   000030  A00
D01     DEVELOPMENT CENTER                   ------  A00
E01     SUPPORT SERVICES                     000050  A00
D11     MANUFACTURING SYSTEMS                000060  D01
D21     ADMINISTRATION SYSTEMS               000070  D01
E11     OPERATIONS                           000090  E01
E21     SOFTWARE SUPPORT                     000100  E01
---------+---------+---------+---------+---------+---------+----
NUMBER OF ROWS DISPLAYED IS 9
STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
SQL CALL FINISHED
```

A

UL/DB2 Internals

In this appendix

- Overview
- How UL/DB2 communicates between Model 204 and DB2
- How UL/DB2 manages threads
- Recovery and restart

Overview

This appendix describes certain aspects of the internal processing done by UL/DB2. Its purpose is to provide background information.

How UL/DB2 communicates between Model 204 and DB2

Before Model 204 can access DB2, UL/DB2 must:

- Establish a connection from Model 204 to DB2
- Create one or more threads on the connection

This connection establishes a communication link between Model 204 and DB2. Within this connection, a thread establishes a two-way path between a Model 204 user and a specific DB2 resource, the application PLAN. Multiple threads can be active within a single connection.

UL/DB2 maintains the thread until the request ends and all changes have been committed or rolled back.

Data and message transfer

Data and status information is passed over these threads between the Model 204 user and DB2.

- Status information generated by DB2 is returned by the User Language $STATUS and $STATUSD functions
- Call Attach Facility messages are returned by the $ERRMSG function
- SQL messages are returned the $DB2EMSG function.

Call Attach Facility

UL/DB2 uses the Call Attach Facility (CAF) to manage the communication between Model 204 and DB2. CAF is an IBM-supplied attachment facility that gives UL/DB2 tight control over the connection between Model 204 and DB2. Using CAF, UL/DB2 monitors and controls the threads connecting Model 204 and DB2. UL/DB2 calls the CAF entry point DSNALI, which gives CAF, and therefore DB2, the appearance of a subroutine.

Accessing DB2

Each time that Model 204 requests access to DB2, UL/DB2 passes program control to DSNALI, a language interface module that is part of CAF. At this point, Model 204 is no longer in control. For this reason, each thread connecting a User Language procedure to DB2 executes under its own Task Control Block (TCB). This TCB is itself a subtask of the Model 204 maintask TCB. This approach avoids having a Model 204 TCB wait for DB2, and provides additional throughput in a multiprocessing environment.

How UL/DB2 manages threads

UL/DB2 provides no system manager commands to control subtasks. This section describes the thread manager component of the UL/DB2 Interface.

After a Model 204 Online initializes, a subtask TCB can be in one of four states:

- Available
- Active, meaning allocated and in use
- Inactive, meaning allocated, but not in use
- Dead, meaning unusable

When a user requests DB2 service, the thread manager assigns the next available subtask TCB to the user task. If no subtasks are available, the thread manager searches for subtasks on the inactive queue.

A TCB is on the inactive queue, because the procedure that initially activated it has ended and issued a CLOSE. Because CONNECT processing is expensive, however, no DISCONNECT is issued. Instead, the subtask control block is placed on the inactive queue as the most recently used control block.

If the user requests DB2 services again, and the subtask control block is available, no CONNECT processing is needed, and the user does not have to be revalidated.

If a new user requests DB2 services and no subtask control blocks are free, the thread manager steals a control block from the inactive chain, starting with the least recently used control block. This “stolen” control block address is removed from the user control block of the last active user. A DISCONNECT is requested for this control block, so that a CONNECT request can be driven to validate the new user ID.

Reusing the same control block is particularly useful for APSY subsystems. Even though a CLOSE is issued at procedure end, the next procedure most probably gets the same subtask control block back, and CONNECT processing can be avoided.

However, if the APSY is defined with an AUTOCOMMIT of NO, the subtask TCB is considered active even through procedure end(s). This is necessary, because the normal CLOSE of a DB2 thread causes a DB2 COMMIT. With an AUTOCOMMIT of NO, commit processing is deferred until the user specifies a Model 204 COMMIT. If there is no Model 204 COMMIT, then a DB2 CLOSE with an ABRT is generated to ensure that DB2 does a ROLLBACK.

Recovery and restart

The Call Attach Facility has no abend recovery routines. Instead, Model 204 provides abend exit routines. These recovery routines use tracking indicators to determine if an abend occurs during DB2 processing.

If an abend occurs while DB2 has control, UL/DB2 issues a CLOSE with the ABRT (abort) option. DB2 detects task termination and terminates the thread. The user loses all database changes since the last COMMIT point, and is also restarted. Any subtask that abends cannot be re-used.

During a user restart, if the user subtask control block has an associated subtask, restart releases the subtask back to the pool of free TCBs. This release terminates the DB2 connection with an ABRT request in order to rollback any uncommitted updates. The DB2 transaction is not atomic with the Model 204 transaction. There is no SYNCPOINT functionality.

Using the ACEE

For security processing, RACHECK processing first checks for an ACEE associated with the TCB. If there is none (TCBSENV = 0), RACHECK then checks the ACEE associated with the Address Space Control Block (ASXBSENV). All subtasks TCBs attached at initialization have the identifier ‘M204’ inserted into the TCB in the TCBUSER field. During CONNECT processing, the requestor’s user ID is copied into an ACEE, and the address of that ACEE is placed in the TCB at TCBSENV for the DB2 authorization exit to use for validation.

Index