Panther

Database Driver–DB2 for UNIX

Release 4.25

Prolifics®

May 2000

1

Database Driver for DB2

This chapter provides documentation specific to DB2. It discusses the following:

- Engine initialization (page 4)
- Connection declaration (page 5)
- Import conversion (page 6)
- Formatting for colon-plus processing and binding (page 9)
- Cursors (page 9)
- Errors and warnings (page 11)
- Database transaction processing (page 14)
- Transaction manager processing (page 16)
- DB2-specific `DBMS` commands (page 17)
- Command directory for Prolifics for DB2 (page 22)

This document is designed as a supplement to information found in the *Developer's Guide*.

Initializing the Database Engine

Database engine initialization occurs in the source file `dbiinit.c`. This source file is unique for each database engine and is constructed from the settings in the `makevars` file. In Prolifics for DB2, this results in the following `vendor_list` structure in `dbiinit.c`:

```
static vendor_t vendor_list[] =
{
   {"db2", dm_db2sup, DM_DEFAULT_CASE ,(char *) 0},

   { (char *) 0, (int (*)()) 0, (int) 0, (char *) 0 }
};
```

The settings are as follows:

`db2`	Engine name. May be changed.
`dm_db2sup`	Support routine name. Do not change.
`DM_DEFAULT_CASE`	Case setting for matching `SELECT` columns with Prolifics variable names. May be changed.

For Prolifics for DB2, the settings can be changed by editing the `makevars.db2` file.

Engine Name

You can change the engine name associated with the support routine `dm_db2sup`. The application then uses that name in `DBMS ENGINE` statements and in `WITH ENGINE` clauses. For example, if you wish to use “tracking” as the engine name, change the following parameter in the `makevars.db2` file:

```
DB2_ENGNAME=tracking
```

If the application is accessing multiple engines, it makes DB2 the default engine by executing:

`DBMS ENGINE` *db2-engine-name*

where *db2-engine-name* is the string used in `vendor_list`. For example,

```
DBMS ENGINE db2
```

or

```
DBMS ENGINE tracking
```

Support Routine Name

`dm_sup` is the name of the support routine for DB2. This name should not be changed.

Case Flag

The case flag, `DM_DEFAULT_CASE`, determines how Prolifics's database drivers use case when searching for Prolifics variables for holding `SELECT` results. This setting is used when comparing DB2 column names to either a Prolifics variable name or to a column name in a `DBMS ALIAS` statement.

DB2 is case insensitive. Regardless of the case in a SQL statement, DB2 creates all database objects—tables, views, columns, etc.—with upper case names. For DB2, the `DM_DEFAULT_CASE` setting is treated as `DM_FORCE_TO_LOWER_CASE`. This means that Prolifics attempts to match DB2 column names to lower case Prolifics variables when processing `SELECT` results. If your application is using this default, use lower case names when creating Prolifics variables.

The case setting can be changed. If you wish to use upper case Prolifics variable names, use the `u` option in the `makevars` file for the `DM_FORCE_TO_UPPER_CASE` flag.

```
DB2_INIT=u
```

If you edit `makevars.db2`, you must remake your Prolifics executables. For more information on engine initialization, refer to Chapter 7 in the *Developer's Guide*.

Connecting to the Database Engine

DB2 allows your application to use one connection at a time. Simultaneous multiple connections are not supported in this release.

The following options are supported for connections to DB2:

Table 1. Database connection options.

Option	Argument
`USER`	*user-name*
`DATABASE`	*database-name*
`PASSWORD`	*password*

Additional keywords are available for other database engines. If those keywords are included in your `DBMS DECLARE CONNECTION` command for DB2, it is treated as an error.

Prolifics always opens the database in shared mode.

You must install and configure the required communications software for DB2 before using Prolifics for DB2. Consult your engine documentation for more information.

Also, DB2 CLI applications require that DB2 CLI bind files are bound to the database. DB2 CLI will auto-bind on the first access to the database, provided the user has the appropriate authorization. The database administrator might need to perform the first connection, or explicitly bind the DB2 CLI bind files and packages. Refer to your CLI reference manual for more information.

Importing Database Tables

The Import⇒Database Objects option in the screen editor creates Prolifics repository entries based on database tables in an DB2 database. When the import process is complete, each selected database table has a corresponding repository entry screen.

In Prolifics for DB2, the following database objects can be imported as repository entries:

- database tables
- database views
- table aliases

After the import process is complete, the repository entry screen contains:

- A widget for each column in the table, using the column’s characteristics to assign the appropriate widget properties.
- A label for each column based on the column name.
- A table view named for the database table, view, or alias.
- Links that describe the relationship between table views.

Each import session allows you to display and select up to 1000 database tables. Each database table can have up to 255 columns. If your database contains more than 1000 tables, use the filter to control which database tables are displayed.

Table Views

A table view is a group of associated widgets on an application screen. As a general rule, the members of a table view are derived from the same database table. When a database table is first imported to a Prolifics repository, the new repository screen has one table view that is named after the database table. All the widgets corresponding to the database columns are members of that table view.

The import process inserts values in the following table view properties:

- Name — The name of the table view, generally the same as the database table.
- Table — The name of the database table.
- Primary Keys — The columns that are defined as primary keys for the database table.
- Columns — A list of the columns in the database table is displayed when you click on the More button. However, this list is for reference only. It cannot be edited.
- Updatable — A setting that determines if the data in the table can be modified. The default setting for Updatable is Yes.

For each repository entry based on a database view, the primary key widgets must be available if you want to update data in that view. To do this, check that the Prolifics table view's Primary Keys property is set to the correct value. Then, the widgets corresponding to the primary keys must be members of either the Prolifics table view or one of its parent table views. For repository entries based on database tables, this information is automatically imported.

Links

Links are created from the foreign key definitions entered in the database. The application screen must contain links if you are using the transaction manager and the screen contains more than one table view.

Check the link properties to see if they need to be edited for your application screen. The Parent and Child properties might need to be reversed or the Link Type might need to be changed.

Refer to Chapter 30 in the *Developer's Guide* for more information on links.

Widgets

A widget is created for each database column. The name of the widget corresponds to the database column name. The Inherit From property is set to `@DATABASE` indicating that the widget was imported from the database engine. The Justification property is set to Left. Other widget properties are assigned based on the data type.

The following table lists the values for the C Type, Length, and Precision properties assigned to each DB2 data type.

Table 2. Importing Database Tables

DB2 Data Type	Prolifics Type	C Type	Widget Length	Widget Precision
`CHAR`	`FT_CHAR`	`Char String`	Column length	
`DATE`	`DT_DATETIME`	`Default`	10	

`DECIMAL`
`NUMERIC`
scale > 0
otherwise
	
	`FT_DOUBLE`
	`FT_INT`
		
		`Double`
		`Int`
			
			Column plus 1
			Column length
				
				Column scale

`FLOAT`	`FT_DOUBLE`	`Double`	16	
`GRAPHIC`	`FT_CHAR`	`Char String`		
`INT`	`FT_LONG`	`Long Int`	11	
`LONG VARCHAR`	`FT_CHAR`	`Char String`	254	
`LONG VARGRAPHIC`	`FT_CHAR`	`Char String`		
`SMALLINT`	`FT_INT`	`Int`	6	
`TIME`	`DT_DATETIME`	`Default`	8	
`TIMESTAMP`	`DT_DATETIME`	`Default`	20	
`VARCHAR`	`FT_CHAR`	`Char String`	Column length; maximum of 254	
`VARGRAPHIC`	`FT_CHAR`	`Char String`		

Scale in DB2 is equivalent to precision in Prolifics.

Other Widget Properties

Based on the column's data type or on the Prolifics type assigned during the import process, other widget properties might be automatically set when importing database tables.

DT_DATETIME

`DT_DATETIME` widgets also have the Format/Display⇒Data Formatting property set to Date/Time and Format Type property set. For `DATETIME` data types, Format Type is set to Default. For `TIME` data types, it is set to Default Time. For `DATE` data types, it is set to Default Date.

Null Field property

If a column is defined to be NOT NULL, the Null Field property is set to No. For example, the roles table in the videobiz database contains three columns: title_id, actor_id and role. title_id and actor_id are defined as NOT NULL so the Null Field property is set to No. role, without a NOT NULL setting, is implicitly considered to allow null values so the Null Field property is set to Yes.

For more information about usage of Prolifics type and C type, refer to Chapter 29 of the *Developer's Guide*.

Formatting for Colon Plus Processing and Binding

This section contains information about the special data formatting that is performed for the engine. For general information on data formatting, refer to Chapter 29 in the *Developer's Guide*.

Formatting Dates

DB2 supports three types of date data types:

- DATE
- TIME
- TIMESTAMP

Declaring Cursors

When a connection is declared to an DB2 engine, Prolifics automatically declares a default cursor for SQL SELECT statements executed with the JPL command DBMS SQL. For all non-SELECT operations performed with DBMS SQL, Prolifics uses DB2's EXECUTE IMMEDIATE feature rather than another default cursor. If the application needs to select multiple rows and update the rows one at a time, the application does not need to declare named cursors.

By default, Prolifics defines 10 cursors for an application accessing DB2. It reserves one for itself (i.e., the "default" cursor); the other nine are available for the application's use. If the application attempts to declare a tenth cursor, Prolifics returns the DM_MANY_CURSORS error. In this case, the application must close a cursor using DBMS CLOSE CURSOR before it can declare a new one. If nine cursors are not enough for your application, please contact JYACC Technical Support.

DB2 supports the use of a select cursor to be used with an associated update or delete operation. In these cases, the cursor should be declared with a `FOR UPDATE OF` clause. For example, you can declare the cursor:

```
DBMS DECLARE cursor1 CURSOR FOR SELECT ... \
   FOR UPDATE OF column-name
```

An update operation could be specified in either an `UPDATE` statement or in another cursor:

```
DBMS SQL UPDATE ... WHERE CURRENT OF cursor1
```

or

```
DBMS DECLARE update_cursor CURSOR FOR UPDATE ... \
   WHERE CURRENT OF cursor1
```

These statements update the last row fetched with `cursor1`. For best results, you need to fetch one row at a time.

The name of the cursor in the `WHERE CURRENT OF` clause must exactly match the name in the cursor declaration, including case.

For more information on cursors, refer to Chapter 27 in the *Developer's Guide*.

Scrolling

Even though DB2 does not have native support for non-sequential scrolling in a select set, Prolifics scrolling is available. Before using any of the following commands:

```
DBMS [ WITH CURSOR cursor-name ] CONTINUE_BOTTOM
```

```
DBMS [ WITH CURSOR cursor-name ] CONTINUE_TOP
```

```
DBMS [ WITH CURSOR cursor-name ] CONTINUE_UP
```

the application must set up a continuation file for the cursor. This is done with this command:

```
DBMS [ WITH CURSOR cursor-name ] STORE FILE [ filename ]
```

To turn off Prolifics scrolling and close the continuation file, use this command:

```
DBMS [ WITH CURSOR cursor-name ] STORE
```

or close the Prolifics cursor with `DBMS CLOSE CURSOR`.

For more information on scrolling, refer to Chapter 28 in the *Developer’s Guide*.

Error and Status Information

Prolifics uses the global variables described in the following sections to supply error and status information in an application. Note that some global variables can not be used in the current release; however, these variables are reserved for use in other engines and for use in future releases of Prolifics for DB2.

Errors

Prolifics initializes the following global variables for error code information:

`@dmretcode`	Standard database driver status code.
`@dmretmsg`	Standard database driver status message.
`@dmengerrcode`	DB2 error code.
`@dmengerrmsg`	DB2 error message.
`@dmengreturn`	Not used in Prolifics for DB2.

DB2 returns error codes and messages when it aborts a command. It usually aborts a command because the application used an invalid option or because the user did not have the authority required for an operation. Prolifics writes DB2 error codes to the global variable `@dmengerrcode` and writes DB2 messages to `@dmengerrmsg`.

Using the Default Error Handler

The default error handler displays a dialog box if there is an error. The first line indicates whether the error came from the database driver or database engine, followed by the text of the statement that failed. If the error comes from the database driver, `Database interface` appears in the `Reported by` list along with the database engine. The error number and message contain the values of `@dmretcode` and `@dmretmsg`. If the error comes from the database engine, only the name of the engine appears in the `Reported by` list. The error number and message contain the values of `@dmengerrcode` and `@dmengerrmsg`.

Using an Installed Error Handler

An installed error or exit handler should test for errors from the database driver and from the database engine. For example:

```
DBMS ONERROR JPL errors
DBMS DECLARE dbi_session CONNECTION FOR ...

proc errors (stmt, engine, flag)
if @dmengerrcode == 0
   msg emsg ”JAM error: ” @dmretmsg
else
   msg emsg ”JAM error: ” @dmretmsg ” %N” \
   ”:engine error is ” @dmengerrcode ” ” @dmengerrmsg
return 1
```

For additional information about engine errors, refer to your DB2 documentation. For more information about error processing in Prolifics, refer to Chapter 36 in the *Developer’s Guide* and Chapter 12 in the *Programming Guide*.

Warnings

Prolifics initializes the following global variables for warning information:

DB2 uses a warning byte called `SQLAWARN` to signal conditions it considers unusual but not fatal. `@dmengwarncode` derives its value from this byte. `@dmengwarncode` is a 10-occurrence array. If DB2 sets a bit in `SQLAWARN`, Prolifics puts a “W” in the corresponding occurrence of `@dmengwarncode`.

The settings for `SQLAWARN` in DB2 are:

Array Index	Meaning
1	Set to W if any of 2 through 10 are set to W. If this is blank, the other fields do not need to be checked.
2	Not applicable in Prolifics for DB2.
3	Set to W if an aggregate function encounters a NULL value.
4	Not applicable in Prolifics for DB2.
5	Set to W if an `UPDATE` or `DELETE` statement does not contain a `WHERE` clause.
6	Set to W if the result of a date calculation was adjusted to avoid an impossible date.
7	Not used.
8	Not used.

Array Index	Meaning
9	Not used.
10	Not used.

Before using `@dmengwarncode`, you should verify these settings for your release of DB2 by consulting your DB2 documentation.

You might wish to use an exit hook function to process warnings. An exit hook function is installed with `DBMS ONEXIT`. A sample exit hook function is shown below.

```
proc check_status (stmt, engine, flag)

if @dmretcode == 0
{
 if @dmengwarncode[1] == "W"
 {
   if @dmengwarncode[3] == "W"
      msg emsg "A NULL value was found."
   if @dmengwarncode[5] == "W"
      msg emsg "The operation did not use a WHERE clause."
 }
}
return 0
```

Row Information

Prolifics initializes the following global variables for row information:

`@dmrowcount`	Count of the number of DB2 rows affected by an operation.
`@dmserial`	Not used in Prolifics for DB2.

DB2 returns a count of the rows affected by an operation. Prolifics writes this value to the global variable `@dmrowcount`.

As explained on the manual page for `@dmrowcount`, the value of `@dmrowcount` after a SQL `SELECT` is the number of rows fetched to Prolifics variables. This number is less than or equal to the total number of rows in the select set. The value of `@dmrowcount` after a SQL `INSERT`, `UPDATE`, or `DELETE` is the total number of rows affected by the operation. Note that this variable is reset when another `DBMS` statement is executed, including `DBMS COMMIT`.

Using Transactions

A transaction is a unit of work that must be totally completed or not completed at all. DB2 has one transaction for each connection. Therefore, in a Prolifics application, a transaction controls all statements executed with a single named connection or the default connection.

The following events commit a transaction on DB2:

- Executing `DBMS COMMIT`.
- Closing the transaction's connection before the transaction is committed.

The following events roll back a transaction on DB2:

- Executing `DBMS ROLLBACK`.

When an application closes a connection with `CLOSE_ALL_CONNECTIONS` or `CLOSE CONNECTION`, DB2 commits any pending transactions on those connections. However, this procedure is not recommended. Instead, it is strongly recommended that applications use explicit `COMMIT` and `ROLLBACK` statements to terminate transactions.

When DB2 commits or rolls back a transaction, it automatically closes all DB2 cursors and removes any prepared statements. Prolifics for DB2 has modified this behavior for a `DBMS COMMIT` statement by declaring all select cursors using the `WITH HOLD` clause. Therefore, fetches can be continued after `DBMS COMMIT`. To begin the fetch after `DBMS ROLLBACK`, the application must simply re-execute the cursor using `DBMS EXECUTE`; it is not necessary to re-declare the Prolifics cursor.

Transaction Control on a Single Connection

After an application declares a connection, a transaction automatically starts on that connection.

DB2 supports the following transaction commands:

- Set availability of autocommit processing.

  ```
  DBMS [WITH CONNECTION connection] AUTOCOMMIT { ON | OFF }
  ```

- Commit the transaction on a default or named connection.

  ```
  DBMS [WITH CONNECTION connection] COMMIT
  ```

- Rollback to the beginning of the transaction on a default or named connection.

  ```
  DBMS [WITH CONNECTION connection] ROLLBACK
  ```

Example

The following example contains a transaction on the default connection with an error handler.

```
# Call the transaction handler and pass it the name
# of the subroutine containing the transaction commands.

call tran_handle ”new_title()”

proc tran_handle (subroutine)
{
# Declare a variable jpl_retcode and
# set it to call the subroutine.
   vars jpl_retcode
   jpl_retcode = :subroutine

# Check the value of jpl_retcode. If it is 0, all statements
# in the subroutine executed successfully and the transaction
# was committed. If it is 1, the error handler aborted the
# subroutine. If it is -1, Prolifics aborted the subroutine.
# Execute a ROLLBACK for all non-zero return codes.

   if jpl_retcode == 0
   {
      msg emsg ”Transaction succeeded.”
   }
   else
   {
      msg emsg ”Aborting transaction.”
      DBMS ROLLBACK
   }
}

proc new_title
   DBMS SQL INSERT INTO titles VALUES \
      (:+title_id, :+name, :+genre_code, \
      :+dir_last_name, :+dir_first_name, :+film_minutes, \
      :+rating_code, :+release_date, :+pricecat)
   DBMS SQL INSERT INTO title_dscr VALUES \
      (:+title_id, :+line_no, :+dscr_text)
   DBMS SQL INSERT INTO tapes VALUES \
      (:+title_id, :+copy_num, :+status, :+times_rented)
DBMS COMMIT
return 0
```

The procedure `tran_handle` is a generic handler for the application’s transactions. The procedure `new_title` contains the transaction statements. This method reduces the amount of error checking code.

The application executes the transaction by executing

```
call tran_handle ”new_title()”
```

The procedure `tran_handle` receives the argument “new_title” and writes it to the variable `subroutine`. It declares a JPL variable, `jpl_retcode`. After performing colon processing, `:subroutine` is replaced with its value, `new_title`, and JPL calls the procedure. The procedure `new_title` begins the transaction, performs three inserts, and commits the transaction.

If `new_title` executes without any errors, it returns 0 to the variable `jpl_retcode` in the calling procedure `tran_handle`. JPL then evaluates the `if` statement, displays a success message, and exits.

If however an error occurs while executing `new_title`, Prolifics calls the application’s error handler. The error handler should display any error messages and return the abort code, 1.

For example, assume the first `INSERT` in `new_title` executes successfully but the second `INSERT` fails. In this case, Prolifics calls the error handler to display an error message. When the error handler returns the abort code 1, Prolifics aborts the procedure `new_title` (therefore, the third `INSERT` is not attempted). Prolifics returns 1 to `jpl_retcode` in the calling procedure `tran_handle`. JPL evaluates the `if` statement, displays a message, and executes a rollback. The rollback undoes the insert to the table `titles`.

Transaction Manager Processing

Transaction Model for DB2

Each database driver contains a standard transaction model for use with the transaction manager. The transaction model is a C program which contains the main processing for each of the transaction manager commands. You can edit this program; however, be aware that the transaction model is subject to change with each release. For DB2, the name of the standard transaction model is `tmdb21.c`.

SAVE Commands

If you specify a `SAVE` command with a table view parameter, it is called a partial command. A partial command is not applied to the entire transaction tree. In the standard transaction models, partial `SAVE` commands do not commit the database transaction. In order to save those changes, you must do an explicit `DBMS COMMIT`. Otherwise, those changes could be rolled back if the database engine performs an automatic rollback when the database connection is closed.

DB2-Specific Commands

Prolifics for DB2 provides commands for DB2-specific features. This section contains a reference page for each command. If you are using multiple engines or are porting an application to or from another engine, please note that these commands may work differently or may not be supported on some engines.

Using Transactions

`AUTOCOMMIT`	Turn autocommit processing on or off.
`COMMIT`	Commit a transaction.
`ROLLBACK`	Rollback a transaction.

AUTOCOMMIT

Turn autocommit transaction processing on or off

`DBMS` *[* `WITH CONNECTION` *connection-name]* `AUTOCOMMIT ON`

`DBMS` *[* `WITH CONNECTION` *connection-name]* `AUTOCOMMIT OFF`

`WITH CONNECTION` *connection-name*	Specify the connection for this command. Because DB2 does not support multiple connections, the `WITH CONNECTION` clause is necessary only in applications using more than one engine.

Description

This command controls whether changes to a database occur immediately upon execution of an `INSERT`, `UPDATE`, or `DELETE` command, or whether they occur when a `DBMS COMMIT` is explicitly executed.

The default setting is `AUTOCOMMIT OFF`. This means that the engine automatically starts a transaction after an application declares a connection. When a recoverable statement (`INSERT`, `UPDATE`, and `DELETE`) is executed, it is not automatically committed. The effects of the statement are not visible until the transaction is terminated. If the transaction is terminated by `DBMS COMMIT`, the updates are committed and visible to other users. If the transaction is terminated by `DBMS ROLLBACK`, the updates are not committed, and the database is restored to its state prior to the start of the transaction. After a transaction is terminated, the engine automatically begins a new transaction.

If the setting is changed to `AUTOCOMMIT ON`, a statement is committed automatically upon successful execution. Its effects are immediately visible to other users, and it cannot be rolled back.

Example

```
proc new_title
   DBMS WITH CONNECTION xxx1 AUTOCOMMIT ON
   call update_title
   msg emsg ”New title data successfully entered.”
   DBMS WITH CONNECTION xxx1 AUTOCOMMIT OFF
return 0
```

```
proc update_title
   DBMS SQL INSERT INTO titles VALUES \
      (:+title_id, :+name, :+genre_code, \
      :+dir_last_name, :+dir_first_name, :+film_minutes, \
      :+rating_code, :+release_date, :+pricecat)
   DBMS SQL INSERT INTO title_dscr VALUES \
      (:+title_id, :+line_no, :+dscr_text)
   DBMS SQL INSERT INTO tapes VALUES \
      (:+title_id, :+copy_num, :+status, :+times_rented)
return 0
```

See Also

`COMMIT`

`ROLLBACK`

COMMIT

Commit a transaction

```
DBMS [ WITH CONNECTION connection-name ] COMMIT
```

`WITH CONNECTION` *connection-name*	Specify the connection for this command. This clause is necessary only in applications using more than one engine because DB2 does not support multiple connections.

Description

Use this command to commit a pending transaction. Committing a transaction saves all the work since the last `COMMIT`. Changes made by the transaction become visible to other users. If the transaction is terminated by `ROLLBACK`, the updates are not committed, and the database is restored to its state prior to the start of the transaction.

This command is available depending on the setting of various parameters in your environment. Refer to the section on transactions and your documentation for more information.

Example

Refer to the example in Using Transactions on page 14.

See Also

Using Transactions on page 14

`ROLLBACK`

ROLLBACK

Roll back a transaction

`DBMS` *[* `WITH CONNECTION` *connection-name]* `ROLLBACK`

`WITH CONNECTION` *connection-name*	This clause is necessary only in applications using more than one engine because DB2 does not support multiple connections.

Description

Use this command to rollback a transaction and restore the database to its state prior to the start of the transaction.

Example

Refer to the example in Using Transactions on page 14.

See Also

Using Transactions on page 14

`COMMIT`

Command Directory for DB2

The following table lists all commands available in Prolifics's database driver for DB2. Commands available to all database drivers are described in the *Programming Guide*.

Table 3. *Commands for DB2*

Command Name	Description	Documentation Location
`ALIAS`	Name a Prolifics variable as the destination of a selected column or aggregate function	*Programming Guide*
`AUTOCOMMIT`	Turn on/off autocommit processing	page 18
`BINARY`	Create a Prolifics variable for fetching binary values	page 810
`CATQUERY`	Redirect select results to a file or a Prolifics variable	
`CLOSE_ALL_CONNECTIONS`	Close all connections on all engines	
`CLOSE CONNECTION`	Close a named connection	
`CLOSE CURSOR`	Close a named cursor	
`COLUMN_NAMES`	Return the column name, not column data, to a Prolifics variable	
`COMMIT`	Commit a transaction	page 20
`CONNECTION`	Set a default connection and engine for the application	
`CONTINUE`	Fetch the next screenful of rows from a select set	*Database Guide* & *Database Drivers*
`CONTINUE_BOTTOM`	Fetch the last screenful of rows from a select set	*Database Guide* & *Database Drivers*
`CONTINUE_DOWN`	Fetch the next screenful of rows from a select set	*Database Guide* & *Database Drivers*

Command Name	Description	Documentation Location
`CONTINUE_TOP`	Fetch the first screenful of rows from a select set	*Database Guide* & *Database Drivers*
`CONTINUE_UP`	Fetch the previous screenful of rows from a select set	*Database Guide* & *Database Drivers*
`DECLARE CONNECTION`	Declare a named connection to an engine	*Database Guide* & *Database Drivers*
`DECLARE CURSOR`	Declare a named cursor	*Database Guide* & *Database Drivers*
`ENGINE`	Set the default engine for the application	
`EXECUTE`	Execute a named cursor	
`FORMAT`	Format the results of a `CAT-QUERY`	
`OCCUR`	Set the number of rows for Prolifics to fetch to an array and set the occurrence where Prolifics should begin writing result rows	
`ONENTRY`	Install a JPL procedure or C function that Prolifics will call before executing a `DBMS` statement	
`ONERROR`	Install a JPL procedure or C function that Prolifics will call when a `DBMS` statement fails	*Database Guide* & *Database Drivers*
`ONEXIT`	Install a JPL procedure or C function that Prolifics will call after executing a `DBMS` statement	
`ROLLBACK`	Roll back a transaction	page 21
`START`	Set the first row for Prolifics to return from a select set	

Command Name	Description	Documentation Location
`STORE`	Store the rows of a select set in a temporary file so the application can scroll through the rows	
`UNIQUE`	Suppress repeating values in a selected column	
`WITH CONNECTION`	Specify the connection to use for a command	
`WITH CURSOR`	Specify the cursor to use for a command	
`WITH ENGINE`	Specify the engine to use for a command	