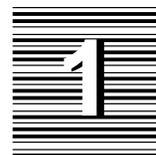


Panther

Database Driver—JDB

Release 4.25



Database Driver for JDB

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

- Engine initialization (page 4)
- Connection declaration (page 5)
- Import conversion (page 6)
- Formatting for colon-plus processing and binding (page 8)
- Cursors (page 8)
- Errors and warnings (page 9)
- Database transaction processing (page 11)
- Transaction manager processing (page 13)
- JDB-specific DBMS commands (page 14)
- Command directory for Prolifics for JDB (page 17)

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 JDB, this results in the following `vendor_list` structure in `dbiinit.c`:

```
static vendor_t vendor_list[] =
{
    {"jdb", dm_jdbsup, DM_DEFAULT_CASE, (char *) 0},
    { (char *) 0, (int (*)()) 0, (int) 0, (char *) 0 }
};
```

The settings are as follows:

<code>jdb</code>	Engine name. May be changed.
<code>dm_jdbsup</code>	Support routine name. Do not change.
<code>DM_DEFAULT_CASE</code>	Case setting for matching <code>SELECT</code> columns with Prolifics variable names. May be changed.

For Prolifics for JDB, the settings can be changed by editing the `makevars.jdb` file.

Engine Name

You can change the engine name associated with the support routine `dm_jdbsup`. 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.jdb` file:

```
JDB_ENGNAME=tracking
```

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

```
DBMS ENGINE jdb-engine-name
```

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

```
DBMS ENGINE jdb
```

or

```
DBMS ENGINE tracking
```

Support Routine Name

`dm_sup` is the name of the support routine for JDB. 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 JDB column names to either a Prolifics variable name or to a column name in a `DEMS ALIAS` statement.

JDB is case insensitive. Regardless of the case in a SQL statement, JDB creates all database objects—tables, views, columns, etc.—with lower case names. For JDB, the `DM_DEFAULT_CASE` setting is treated as `DM_FORCE_TO_LOWER_CASE`. Because JDB uses only lower case, the `DM_FORCE_TO_LOWER_CASE` setting is the same as `DM_PRESERVE_CASE`. For either of these flags, Prolifics attempts to match JDB 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.

If you wish to use upper case variable names, substitute the `u` option in the `makevars` file that sets the `DM_FORCE_TO_UPPER_CASE` flag.

```
JDB_INIT=u
```

If you edit `makevars.jdb`, 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

JDB allows your application to use one or more connections. The application can declare any number of named connections with `DEMS DECLARE CONNECTION` statements; however, you should not have multiple connections to the same database.

The following options are supported for connections to JDB:

Table 1. Database connection options.

Option	Argument
<code>DATABASE</code>	<i>database-pathname</i>

database-pathname is a pathname to an existing database.

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

Importing Database Tables

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

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.
- 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.

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 JDB data type.

Table 2. *Importing Database Tables*

JDB Data Type	Prolifics Type	C Type	Widget Length	Widget Precision
char	FT_CHAR	Char String	Column length	
datetime	DT_DATETIME	Default	20	
double	FT_FLOAT	Double	16	2
float	FT_FLOAT	Float	16	2
int	FT_LONG	Long Int	11	
long	FT_LONG	Long Int	11	

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 set to DEFAULT. Note that dates in this Format Type appear as:

MM/DD/YY HH:MM

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*.

Declaring Cursors

Prolifics uses two cursors for operations performed by DBMS SQL. One cursor is used for SQL SELECT statements and the other for non-SELECT statements. These two cursors might be sufficient for small applications. Larger applications often require more; an application might declare named cursors using DBMS DECLARE CURSOR. For example, master and detail applications often need to declare at least one named cursor: one cursor selects the master rows and additional cursors select detail rows. In short, if an application is processing a SELECT set in increments (i.e., by using DBMS CONTINUE) while it is executing other SELECT statements, two or more cursors are necessary.

Prolifics does not put any limit on the number of cursors an application may declare to an JDB engine. Because each cursor requires memory and JDB resources, however, it is recommended that applications close a cursor when it is no longer needed.

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

Scrolling

Even though JDB 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 JDB.

Errors

Prolifics initializes the following global variables for error code information:

@dmretcode	Standard database driver status code.
@dmretmsg	Standard database driver status message.
@dmengerrcode	JDB error code.
@dmengerrmsg	JDB error message.
@dmengwarncode	Not used in Prolifics for JDB.
@dmengwarnmsg	Not used in Prolifics for JDB.
@dmengreturn	Not used in Prolifics for JDB.

JDB 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 JDB error codes to the global variable @dmengerrcode and writes JDB messages to @dmengerrmsg.

All JDB errors are Prolifics errors. Therefore, Prolifics always calls the default error handler or the installed error handler when an error occurs.

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 JDB 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*.

Row Information

Prolifics initializes the following global variables for row information:

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

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. JDB 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 JDB:

- Executing `DBMS COMMIT`.
- Closing the connection.

The following events roll back a transaction on JDB:

- Executing `DBMS ROLLBACK`.

Transaction Control on a Single Connection

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

JDB supports the following transaction commands:

- 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 JDB

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 JDB, the name of the standard transaction model is `tmjdb1.c`.

Even though JDB does not enforce referential integrity, the transaction manager checks for duplicate primary key values each time data is inserted or updated. This is performed through processing found in the standard transaction model for JDB. If it finds any duplicate value in the primary key columns, the transaction manager gives an error.

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.

JDB-Specific Commands

Prolifics for JDB provides commands for JDB-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

COMMIT	Commit a transaction.
ROLLBACK	Rollback a transaction.

COMMIT

Commit a transaction

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

WITH CONNECTION *connection-name* Specify the connection for this command. If the command does not contain a WITH CONNECTION clause, Prolifics issues the commit on the default connection.

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.

After a transaction is terminated, the engine automatically begins a new transaction.

When an application closes its connections with `CLOSE_ALL_CONNECTIONS` or `CLOSE CONNECTION`, JDB 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.

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 11.

See Also

Using Transactions on page 11

`ROLLBACK`

ROLLBACK

Roll back a transaction

DBMS [*WITH CONNECTION connection-name*] ROLLBACK

WITH CONNECTION connection-name Specify the connection for this command. If the command does not contain a *WITH CONNECTION* clause, Prolifics issues the rollback on the default connection.

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

If a statement in a transaction fails, an application must attempt to reissue the statement successfully or else roll back the transaction. If an application cannot complete a transaction, it should roll back the transaction. If it does not, it might inadvertently commit the partial transaction when it commits a later transaction.

Example Refer to the example in Using Transactions on page 11.

See Also Using Transactions on page 11

COMMIT

Command Directory for JDB

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

Table 3. *Commands for JDB*

Command Name	Description	Documentation Location
ALIAS	Name a Prolifics variable as the destination of a selected column or aggregate function	<i>Programming Guide</i>
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 15
CONNECTION	Set a default connection and engine for the application	
CONTINUE	Fetch the next screenful of rows from a select set	<i>Database Guide & Database Drivers</i>
CONTINUE_BOTTOM	Fetch the last screenful of rows from a select set	<i>Database Guide & Database Drivers</i>
CONTINUE_DOWN	Fetch the next screenful of rows from a select set	<i>Database Guide & Database Drivers</i>
CONTINUE_TOP	Fetch the first screenful of rows from a select set	<i>Database Guide & Database Drivers</i>

Command Name	Description	Documentation Location
CONTINUE_UP	Fetch the previous screenful of rows from a select set	<i>Database Guide & Database Drivers</i>
DECLARE CONNECTION	Declare a named connection to an engine	<i>Database Guide & Database Drivers</i>
DECLARE CURSOR	Declare a named cursor	<i>Database Guide & Database Drivers</i>
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	<i>Database Guide & Database Drivers</i>
ONEXIT	Install a JPL procedure or C function that Prolifics will call after executing a DBMS statement	
ROLLBACK	Roll back a transaction	page 16
START	Set the first row for Prolifics to return from a select set	
STORE	Store the rows of a select set in a temporary file so the application can scroll through the rows	

Command Name	Description	Documentation Location
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	

