Panther

Database Driver–ODBC

Release 4.25

Prolifics®

May 2000

Database Driver for ODBC

ODBC (Open Database Connectivity) defines a library of function calls and SQL syntax based on the X/Open and SQL Access Group specification. It provides application builders with a standard programming interface, standard set of error codes, standard way to connect to a DBMS, and a standard repository of data types.

The ODBC architecture has four components:

- Application — Calls ODBC functions to submit SQL statements and fetch results. This includes Prolifics, Prolifics's ODBC driver, and the applications screens, JPL scripts, and menus.
- Driver Manager — Loads ODBC drivers for an application. This software is usually supplied by Microsoft or Visigenics. On Windows, Microsoft supplies a dynamically linked library `ODBC.DLL`. This software is not supplied with Prolifics.
- ODBC Driver— Processes ODBC function calls, submits SQL statements to a data source, and fetches results to an application. The ODBC driver is supplied by any of a number of third–party vendors. In some cases, the database vendor might supply an ODBC driver. Other companies, such as Intersolv, supply a package of drivers for several DBMS products. This software is not supplied with Prolifics.

- Data source — Comprises the data and its operating system, DBMS, and any network software. For example, a data source might be a local xBase file, a SYBASE RDMBS running on Unix workstation accessed by TCP/IP, or an Oracle RDBMS running on Windows NT accessed by Windows sockets. This software is not supplied with Prolifics.

The ODBC API defines a set of core functions that correspond to the functions in the X/Open and SQL Access Group CLI (Call Level Interface) specification. ODBC also defines two sets of extended functions, Level 1 and Level 2. Unless otherwise documented, Prolifics functions use ODBC core functions. If Prolifics requires a Level 1 or 2 function for some feature, the application's ODBC driver must support the function to use the feature.

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

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

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

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

The settings are as follows:

odbc	Engine name. May be changed.
dm_odbsup	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 ODBC, the settings can be changed by editing the makevars.odb file.

Engine Name

You can change the engine name associated with the support routine dm_odbsup. 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.odb file:

```
ODB_ENGNAME=tracking
```

Using ODBC, your application can access multiple DBMS products. However, in such cases, Prolifics views the application as accessing one database engine, odbc. The information to access each of the subsequent database engines is set in the DBMS DECLARE CONNECTION statement using the DATASOURCE keyword. For more information, refer to page 6.

Support Routine Name

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

When the case flag is set to `DM_DEFAULT_CASE`, Prolifics for ODBC tests for the value of `SQL_IDENTIFIER_CASE` using the ODBC `SQLGetInfo` function. For case sensitive engines, Prolifics then sets the case flag to `DM_PRESERVE_CASE`. This matches the engine column name to a Prolifics variable of the same name and case when processing `SELECT` results. For case insensitive engines, it sets the case flag to `DM_FORCE_TO_LOWER_CASE`. This means that Prolifics attempts to match the engine 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.

```
ODB_INIT=u
```

If you edit `makevars.odb`, 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

ODBC allows your application to use one or more connections. The application can declare any number of named connections with `DBMS DECLARE CONNECTION` statements, up to the maximum number permitted by the ODBC driver and data source.

To access multiple database engines using ODBC, there needs to be a data source for each database engine. Refer to your ODBC database driver documentation for additional information.

The following options are supported for connections to ODBC:

Table 1. Database connection options.

Option	Argument	Conformance Level
`USER`	*user-name*	core
`PASSWORD`	*password*	core
`DATABASE`	*database-name*	1 using `SQLDriverConnect`
`DATASOURCE`	*data-source-name*	core

Option	Argument	Conformance Level
CONN_STRING	*connection-parameters*	1 using SQLDriverConnect
COMPLETION	*connection–mode*	1 using SQLDriverConnect

The USER, PASSWORD and DATASOURCE options are supported by all ODBC database drivers. The CONN_STRING and COMPLETION options are available if the ODBC driver has Level 1 conformance. The DATABASE option is only available with certain ODBC drivers. If you are unsure of the driver's conformance level, use only the core conformance arguments.

DATASOURCE specifies the data source entered in the ODBC Administrator to use for connecting to the database. This data source entry typically contains the name of the data source, the virtual node, and the full path of the database files. A data source is created with the ODBC utility ODBCADM.

The application must supply the DATASOURCE using this flag or by prompting the user with a COMPLETION dialog.

Some drivers support or require additional logon arguments. The program might supply them with the argument CONN_STRING. Alternately, the application might prompt the user for the data using the dialogs of the ODBC driver manager and the ODBC database driver. The connection flag COMPLETION determines whether or not dialogs are used.

CONN_STRING allows you to enter any number of driver-defined keywords and values. The format for the connection string is:

"keyword=value;keyword=value"

If, for example, the driver supports the attribute MS to determine whether the driver modifies SQL statements to conform to ODBC specifications and the attribute LANG to specify national language, the CONN_STRING argument is:

```
CONN_STRING "MS=1;LANG=FRENCH"
```

Consult your ODBC driver documentation about the supported connection attributes for your database. Note that Prolifics does not attempt to validate the CONN_STRING value.

COMPLETION specifies the mode used by the Driver Manager and the ODBC database driver to establish a connection to a data source. The mode can be set to any of the following:

COMPLETE	NOPROMPT
COMPLETE_REQUIRED	PROMPT

For PROMPT, the ODBC Driver Manager always initiates a dialog box containing the installed data source names and prompts for information.

For COMPLETE, the ODBC Driver Manager initiates a dialog box only if there is not enough information in the connection string to connect to the data source.

COMPLETE_REQUIRED is similar to COMPLETE. The ODBC Driver Manager initiates a dialog box only if there is not enough information in the connection string to connect to the data source. It also grays and disables any prompts on the dialog that are not required.

For NOPROMPT, the ODBC Driver Manager attempts to connect to the data source and does not display a dialog box. NOPROMPT is the default.

Prolifics for ODBC also supports the argument DATABASE. This argument corresponds to the connection attribute DB. Driver vendors, such as Intersolv, often use DB to supply the database name. If your driver supports the DB attribute, the application can set it using the CONN_STRING keyword:

```
DECLARE c1 CONNECTION FOR DATASOURCE "SYB49" \
   CONN_STRING "DB=pubs2"
```

or with the DATABASE keyword:

```
DECLARE c1 CONNECTION FOR DATASOURCE "SYB49" \
   DATABASE "pubs2"
```

If the Driver Manager finds the data source specification, it loads the driver. If the Driver Manager cannot find the data source specification and if there is no default specification, it returns an error.

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

If you get the error message "Login Denied" when you issue the connection statement, check the data source name. This message is issued when the data source name is invalid.

Importing Database Tables

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

The Prolifics importer requires the ODBC catalog functions:

- `SQLTables` — Level 1
- `SQLColumns` — Level 1

If these functions are not supported, the importer will fail.

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

- database tables
- database views
- synonyms

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, database table view, or synonym.
- 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 or unique indexes for the database table. The importer calls `SQLPrimaryKeys` or

`SQLStatistics` to find a primary key. If the ODBC driver does not support either function, the importer cannot set this property.

- 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.

If the database engine does not support foreign key definitions or if the ODBC driver does not support `SQLForeignKeys`, the links needed by the transaction manager will have to be created manually if the application screen contains more than one table view.

If you are using the screen wizard to create screens, the links must also be added to the repository entries in order for the wizard to allow more than one table view in each section of a screen.

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 ODBC data type.

Table 2. Importing Database Tables

ODBC Data Type	Code	Prolifics Type	C Type	Widget Length	Widget Precision
SQL_BIGINT	–5	FT_LONG	Long Int	column length + 1	
SQL_BINARY	–2	DT_BINARY	Hex Dec	column length * 2	
SQL_BIT	–7	FT_INT	Int	column length + 1	
SQL_CHAR	1	FT_CHAR	Char String	column length	
SQL_DATE	9	DT_DATETIME	Default	20	
SQL_DECIMAL	3				
(ODBC scale = 0)		FT_INT	Int	column length	
(ODBC scale > 0)		FT_DOUBLE	Double	column length + 2	Same as column scale
SQL_DOUBLE	8	FT_DOUBLE	Double	22	2
SQL_FLOAT	6	FT_DOUBLE	Double	22	2
SQL_INTEGER	4	FT_LONG	Long Int	column length + 1	
SQL_LONGVARBINARY	–4	DT_BINARY	Hex Dec	column length * 2	
SQL_LONGVARCHAR	–1	FT_CHAR	Char String	column length	
SQL_NUMERIC	2				
(ODBC scale = 0)		FT_INT	Int	column length	
(ODBC scale > 0)		FT_DOUBLE	Double	column length + 2	Same as column scale
SQL_REAL	7	FT_FLOAT	Float	13	
SQL_SMALLINT	5	FT_INT	Int	column length + 1	
SQL_TIME	10	DT_DATETIME	Default	20	
SQL_TINYINT	–6	FT_INT	Int	column length + 1	
SQL_TIMESTAMP	11	DT_DATETIME	Default	20	

ODBC Data Type	Code	Prolifics Type	C Type	Widget Length	Widget Precision
SQL_VARBINARY	–3	DT_BINARY	Hex Dec	column length * 2	
SQL_VARCHAR	12	FT_CHAR	Char String	column length	

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.

UseInUpdate property

If a column's length is defined as larger than 254 in the database, then the database importer sets the Use In Update property to No for the widget corresponding to that column. Because widgets in Prolifics have a maximum length of 254, the data originally in the database column could be truncated as part of a SAVE command in the transaction manager.

The Use In Update property is also set to No for certain data types, such as the timestamp column in SYBASE.

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*.

Formatting Dates

Prolifics uses the ODBC standard formats for converting Prolifics date values to valid ODBC values. If the Prolifics widget has date and time edits, it is formatted

as a timestamp; if the widget has only time edits, it is formatted as a time; and if the widget has only date edits, it is formatted as a date. The formats are:

Column Type	Format
Date	{d 'YYYY:MM:DD'}
Time	{t 'HH:MM:SS'}
Timestamp	{ts 'YYYY:MM:DD HH:MM:SS'}

where YYYY, MM, DD, HH, MM, and SS are specified as integers.

Declaring Cursors

When a connection is declared to an ODBC 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 ODBC's SQLExecDirect function rather than another default cursor. This feature is also known as EXECUTE IMMEDIATE. 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.

If the driver is unable to perform the operation using SQLExecDirect, Prolifics returns the error DM_CANNOT_EXEC_IMMED. In this case, the application should declare and execute a named cursor for the operation.

Applications should also use a named cursor to execute a catalog function or a stored procedure.

Prolifics does not put any limit on the number of cursors an application may declare to an ODBC engine. Because each cursor requires memory and ODBC 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 ODBC 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 ODBC.

Errors

Prolifics initializes the following global variables for error code information:

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

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

All ODBC 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 ODBC 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 ODBC rows affected by an operation.

ODBC 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 Stored Procedures

A stored procedure is a precompiled set of SQL statements that are recorded in the database and executed by calling the procedure name. Since the SQL parsing and syntax checking for a stored procedure are performed when the procedure is created, executing a stored procedure is faster than executing the same group of SQL statements individually. By passing parameters to and from the stored procedure, the same procedure can be used with different values. In addition to SQL statements, stored procedures can also contain control flow language, such as `if` statements, which gives greater control over the processing of the statements.

The syntax for executing a stored procedure in ODBC is:

```
DBMS DECLARE cursor-name CURSOR FOR \
    { CALL procedure-name [  (::parameter [,  ::[ parameter ]...) ]}
```

In the current release of Prolifics for ODBC, an application cannot execute a stored procedure containing output parameters or return codes. Normal select result sets are supported.

Using Transactions

A transaction is a unit of work that must be totally completed or not completed at all. ODBC 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 ODBC:

- Executing `DBMS COMMIT`.

The following events roll back a transaction on ODBC:

- Executing `DBMS ROLLBACK`.

Transactions are not available for all database drivers using ODBC. Refer to your ODBC database driver documentation for more information.

Transaction Control on a Single Connection

ODBC 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`

The setting for autocommit processing also determines the availability of other transaction commands. If the setting is `AUTOCOMMIT ON`, every statement is committed immediately. The other transaction commands—`COMMIT`, `ROLLBACK`—are invalid. If the setting is `AUTOCOMMIT OFF`, the statements in a transaction must be committed in order for the work to be saved and visible to the rest of the application or other users. `AUTOCOMMIT ON` is the default setting for drivers that support this feature.

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 ODBC

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 ODBC, the name of the standard transaction model is `tmodb1.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.

ODBC-Specific Commands

Prolifics for ODBC provides commands for ODBC-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.

Retrieving System Information

`DECLARE CURSOR FOR CATALOG_FUNCTION`	Declare a cursor for retrieving system information.

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. If this clause is not included, Prolifics issues the command on the default connection.

Environment

Some ODBC drivers and data sources do not support this command. This command requires a level 1 conformance function, SQLSetConnectOption.

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.

Prolifics sets the value to 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. If the command does not contain a `WITH CONNECTION` clause, Prolifics issues the commit on the default connection.

Environment

Some ODBC drivers and data sources do not support this command.

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 16.

See Also

Using Transactions on page 16

`ROLLBACK`

DECLARE CURSOR FOR CATALOG_FUNCTION

Declare a named cursor for an ODBC system catalog function

```
DBMS [ WITH CONNECTION connection-name ] DECLARE cursor-name CURSOR \
       FOR CATALOG_FUNCTION function-name [ ::parameter [ ::parameter  ... ] ]
```

`WITH CONNECTION` *connection-name*
: Specify the connection for this command. If this clause is not included, Prolifics associates the cursor with the default connection.

function-name
: Specify the name of the ODBC function. The name is not case-sensitive. Supported functions include:

SQLColumns	SQLPrimaryKeys	SQLStatistics
SQLColumnPrivileges	SQLProcedures	SQLTablePrivileges
SQLForeignKeys	SQLProcedureColumns	SQLTables
SQLGetTypeInfo	SQLSpecialColumns	

parameter
: Specify a valid parameter name for the function. The parameter must begin with a double colon, which is the Prolifics syntax for cursor parameters.

Environment

Some ODBC drivers and data sources do not support this command.

Description

Use this command to create a named cursor to call an ODBC function and retrieve information from the system catalog. The keyword `CATALOG_FUNCTION` is required. Following the keyword are the name of the function and the function's parameters. For more information on each function, including the function's parameters, refer to your ODBC documentation.

Example

```
DBMS DECLARE x CURSOR FOR CATALOG_FUNCTION sqltables \
    ::parm1 ::parm2 ::parm3 ::parm4

DBMS WITH CURSOR x EXECUTE USING '', '%', '%', ''
```

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.

Environment

Some ODBC drivers and data sources do not support this command.

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 16.

See Also

Using Transactions on page 16

`COMMIT`

Command Directory for ODBC

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

Table 3. Commands for ODBC

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 20
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 22
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*
DECLARE CURSOR FOR CATALOG_FUNCTION	Declare a cursor to execute an ODBC catalog function	page 23
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 24
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	

Library Functions for ODBC

Prolifics for ODBC provides an additional C function in order to obtain the connection information. This function is described in this section.

dm_odb_get_dbhandle

Get the current connection handle

```
#include <dmodbsup.h>

HDBC dm_odb_get_dbhandle(char *connection);
```

connection	Prolifics for ODBC connection name.

Returns

- If connection is valid, return associated HDBC.
- If connection is NULL or an empty string, return HDBC of the current connection.
- Otherwise, return SQL_NULL_HDBC.

Description

dm_odb_get_dbhandle returns the ODBC connection handle (HDBC) for the named Prolifics connection. This handle is needed if you wish to call ODBC SDK functions, such as SQLGetInfo.

The Prolifics for ODBC distribution includes a sample file that is located in $SMBASE\ODBC\ODBCSAMP.C. It defines some sample functions that use dm_odb_get_dbhandle. To call these sample functions from JPL or from control strings, copy the sample file to your working directory, add the file name to the SRCMODS macro in the makevars.odb file, install the functions in the prototyped function list, and rebuild the executable. For more information about installing functions in the prototyped function list, refer to Chapter 43 in the *Application Development Guide*.

Example

```
#include <smdefs.h>
#include <dmodbsup.h>

SWORD
sm_odbinfo (connection, flag)
char *connection;
UWORD flag;
{
    HDBC      dbhandle;
    SWORD     value;
    RETCODE   retcode;
```

```
    dbhandle = dm_odb_get_dbhandle(connection);

    if (dbhandle != SQL_NULL_HDBC)
    {
        retcode = SQLGetInfo(dbhandle, flag, (PTR)&value,
              sizeof(value), NULL);
        if (retcode == SQL_SUCCESS)
        {
            return value;
        }
    }
    return -1;
}
```

The following example is in JPL and it assumes you have installed `sm_odbinfo` in the function list:

```
# include <odbcgbls>

vars cursor_stat(5)
cursor_stat=sm_odbinfo("dm_odb_0conn", \
       SQL_CURSOR_COMMIT_BEHAVIOR)
if (cursor_stat < 2)
{
    # Cursors are closed after commit. Application must
    # re-execute SELECT cursors.
}
else
{
    # Cursors remain open after commit. Application may
    # call CONTINUE.
}
```