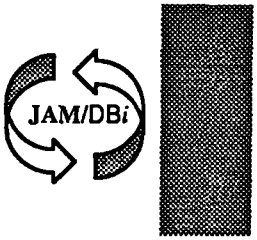
JAM/DB*i* for ORACLE

August 17, 1992



Notes for ORACLE

This appendix provides documentation specific to ORACLE.

It discusses the following:

- engine initialization
- connection declaration
- cursors
- formatting for colon-plus and binding
- errors and warnings
- utilities
- engine-specific features
- command directory for JAM/DB*i* ORACLE

This document is designed as a supplement, not a replacement, to the JAM/DB*i* manual. Each section identifies its companion chapter or section in the JAM/DB*i* manual.

1.1 ENGINE INITIALIZATION

See Section 7.1

By default, JAM/DB*i* uses the following values in `dbiinit.c` for ORACLE initialization:

```
static vendor_t vendor_list[] =
{
    {"oracle", dm_orasup, DM_FORCE_TO_LOWER_CASE ,(char *) 0},

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

The default settings are as follows:

`oracle`	Engine name. May be changed.
`dm_orasup`	Support routine name. Do not change.
`DM_FORCE_TO_LOWER_CASE`	Case setting for matching `SELECT` columns with JAM variable names. May be changed.

1.1.1
Engine Name and Support Routine

An application may change the engine name associated with the support routine `dm_orasup`. 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, make the following change:

```
static vendor_t vendor_list[] =
{
    {"tracking", dm_orasup,DM_FORCE_TO_LOWER_CASE,(char *) 0},

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

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

```
dbms ENGINE oracle_engine_name
```

where *oracle_engine_name* is the string used in `vendor_list`. For example,

```
dbms ENGINE oracle
```

or

```
dbms ENGINE tracking
```

`dm_orasup` is the name of the support routine for ORACLE. The support routine uses ORACLE's Call Interface (OCI). This name should not be changed.

If your application is using multiple engines, you need to add a line to `vendor_list` for each engine. You also need to modify your makefile to support both engines and recompile the JAM/DB*i* executables, `jxdbi` and `jamdbi`.

1.1.2
Case and Error Flags

The case flag, `DM_FORCE_TO_LOWER_CASE`, determines how JAM/DB*i* uses case when searching for JAM variables for holding `SELECT` results. JAM/DB*i* uses this setting when

comparing ORACLE column names to either a JAM variable name or to a column name in a DBMS ALIAS statement.

ORACLE is case insensitive. Regardless of the case in a SQL statement, ORACLE creates all database objects—tables, views, columns, etc.,—with upper case names. In SQL statements, users may use any case to refer to these objects. By default, JAM/DB*i* initializes case-insensitive engines using the DM_FORCE_TO_LOWER_CASE flag. This means that JAM/DB*i* attempts to match an ORACLE column name to a lower case JAM variable name when processing SELECT results. If your application is using this default, use lower case names when creating JAM variables.

The case setting may be changed. If you wish to use upper case JAM variable names, replace DM_FORCE_TO_LOWER_CASE with DM_PRESERVE_CASE or DM_FORCE_TO_UPPER_CASE.

You may also set an optional flag to change the behavior of JAM/DB*i*'s default error handler. An application may set either of the following:

DM_DEFAULT_DBI_MSG	Set the default error handler to display standard JAM/DB*i* messages for all error messages.
DM_DEFAULT_ENG_MSG	Set the default error handler to display ORACLE error messages instead of JAM/DB*i* error messages.

If neither flag is used, DM_DEFAULT_DBI_MSG is the default. To show ORACLE error messages as the default, use the bitwise OR operator and DM_DEFAULT_ENG_MSG:

```
static vendor_t vendor_list[] =
{
    {"oracle", dm_orasup, DM_FORCE_TO_LOWER_CASE | DM_DEFAULT_ENG_MSG,
    (char *) 0 },

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

If you modify the settings in `dbiinit.c`, you must recompile and link the JAM/DB*i* executables, `jxdbi` and `jamdbi`. `dbiinit.c` does not affect the utility executables, `tbl2f` and `f2tbl`.

Please note that DM_DEFAULT_DBI_MSG and DM_DEFAULT_ENG_MSG do not affect an application using an error hook function. An error hook function is installed with DBMS ONERROR and controls all error message display.

1.2 CONNECTION

See Section 7.2

The following options are supported for connections to ORACLE:

USER	***user_name***
PASSWORD	***password***

where ***user_name*** is a valid logon name for the ORACLE database. ***user_name*** must have CONNECT privileges to logon. For more information see your DBA or the ORACLE RDBMS Database Administrator's Guide.

The syntax is,

```
dbms [WITH ENGINE engine] DECLARE connection CONNECTION \
   [FOR USER user_name [PASSWORD password] ]
```

For example,

```
dbms DECLARE dbi_session CONNECTION FOR \
   USER :+uname PASSWORD :+pword
```

where `uname` and `pword` are JAM field names.

ORACLE allows your application to use one or more connections. The application may declare any number of named connections with `DBMS DECLARE CONNECTION` statements.

1.3 CURSORS

See Section 7.3

JAM/DB*i* uses two cursors for operations performed by `sql` and its equivalents, `dm_sql` and `dm_sql_noexp`. In ORACLE terminology, a cursor is also known as a *context area*. JAM/DB*i* uses one cursor for `SELECT` statements and the other for non-`SELECT` statements. These two cursors may be sufficient for small applications. Larger applications often require more; an application may 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.

Declaring a named cursor may improve the performance of some `SELECT` statements. In particular, if an application is executing a `SELECT` statement more than once and the `SELECT` fetches 40 or more columns from a remote server, a named cursor is recommended. In this

case, the parse and describe is done just once when the cursor is declared, not each time the cursor is executed.

JAM/DB*i* does not put any limit on the number of cursors an application may declare to an ORACLE engine. Since each cursor requires memory and ORACLE resources, however, it is recommended that applications close a cursor when it is no longer needed.

1.4 FORMATTING FOR COLON-PLUS AND BINDING

See Chapter 8

JAM/DB*i* uses ORACLE's built-in `TO_DATE` function and the ORACLE format string, `ddmmyyyy hh24miss` to convert JAM dates to ORACLE form.

1.5 SCROLLING

See Section 9.1.2

ORACLE does not have native support for backward scrolling in a `SELECT` set. Before using any of the following commands

```
dbms [WITH CURSOR cursor] CONTINUE_BOTTOM
dbms [WITH CURSOR cursor] CONTINUE_TOP
dbms [WITH CURSOR cursor] CONTINUE_UP
```

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

```
dbms [WITH CURSOR cursor] STORE FILE [filename]
```

1.6 ERROR AND STATUS INFORMATION

See Section 9.2 and Chapter 13

In Release 5, JAM/DB*i* uses the global variables described in the following sections to supply error and status information in an application. Note that some global variables may not be used in the current release; however, these variables are reserved for use in other engines and for use in future releases of JAM/DB*i* for ORACLE.

1.6.1
Errors

JAM/DB*i* initializes the following global variables for error code information:

`@dmretcode`	Standard JAM/DB*i* status code.
`@dmretmsg`	Standard JAM/DB*i* status message.
`@dmengerrcode`	ORACLE error code.
`@dmengerrmsg`	ORACLE error message.
`@dmengreturn`	Not used in JAM/DB*i* for ORACLE.

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

All ORACLE errors are JAM/DB*i* errors. Therefore, JAM/DB*i* always calls the default or the installed error handler when an error occurs.

The easiest way to test for ORACLE errors is with an installed error or exit handler. For example,

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

proc errors
parms stmt engine flag
   if @dmengerrcode == 0
      msg emsg "JAM/DBi error: " @dmretmsg
   else
      msg emsg "JAM/DBi error: " @dmretmsg " %N" \
      ":engine error is" @dmengerrcode " " @dmengerrmsg
   return 1
```

If you need additional information about ORACLE errors, please consult your ORACLE documentation.

1.6.2
Warnings

JAM/DB*i* initializes the following global variables for warning information:

`@dmengwarncode`	ORACLE bit warning flag.
`@dmengwarnmsg`	Not used in JAM/DB*i* for ORACLE.

ORACLE uses a warning byte called `flags1` to signal conditions it considers unusual but not fatal. `@dmengwarncode` derives its value from this byte. `@dmengwarncode` is an 8–occurrence array. If ORACLE sets a bit in `flags1` of the Cursor Data Area, JAM/DB*i* puts a "W" in the corresponding occurrence of `@dmengwarncode`. The settings for `flags1` in ORACLE 6.0 are:

Bit Value	*Meaning*
001	There is a warning. This is set when any other bit in `flags1` is set.
002	Set if any data item was truncated.
003	Unused.
004	Unused
005	Set if an `UPDATE` or `DELETE` statement does not contain a `WHERE` clause.
006	Unused.
007	Unused.
008	Unused.

Before using `@dmengwarncode`, you should verify these settings by consulting your *Oracle Call Interfaces Manual*.

You may 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
parms stmt engine flag

if @dmretcode == 0
{
    if @dmengwarncode [1] == "W"
    {
        if @dmengwarncode [2] == "W"
            msg emsg "Some data was truncated."
```

```
        else if @dmengwarncode [5] == "W"
           msg emsg "The operation did not use a WHERE clause."
    }
}
return
```

1.6.3

Row Information

JAM/DB*i* initializes the following global variables for row information:

`@dmrowcount`	Count of the number of ORACLE rows affected by an operation.
`@dmserial`	Not used in JAM/DB*i* for ORACLE.

ORACLE returns a count of the rows affected by an operation. JAM/DB*i* writes this value to the global variable `@dmrowcount`.

As explained on the manual page for `@dmrowcount`, the value of `@dmrowcount` after a `SELECT` is the number of rows fetched to JAM variables which may be less than or equal to the total number of rows in the select set. Immediately after an `INSERT`, `UPDATE`, or `DELETE`, `@dmrowcount` is set to the total number of rows affected by the operation. This variable is cleared whenever a `DBMS COMMIT` statement is executed.

1.7

UTILITIES

See Chapter 16

If you start the utilities in interactive mode using the `-i` flag, the utility displays an engine-independent logon screen. JAM/DB*i* uses the following options:

- `User`
- `Password`

when declaring a connection to ORACLE for the utilities. Enter the same information you use to declare a connection in `jamdbi`. The other fields on the logon screen may remain empty.

1.7.1

f2tbl

`f2tbl` creates a database table based on a JAM form. It uses each named field on the form to create a column, translating field edits to an appropriate ORACLE column definition. The table below shows the default ORACLE column definitions for each JAM type.

If you do not know how to check a field's JAM type, please see the *Utility Reference Chapter* of the JAM/DB*i* manual.

JAM *Type*	*ORACLE Column Definition*		
	Type	*Length*	*Precision*
DT_CURRENCY	NUMBER	Same as field length (maximum of 42)	Field's precision[1]
DT_DATETIME	DATE		
DT_YESNO	CHAR	Same as field length (maximum of 240)	
FT_CHAR	CHAR	Same as field length (maximum of 240)	
FT_DOUBLE	NUMBER	Same as field length (maximum of 42)	Field's precision[1]
FT_FLOAT	NUMBER	Same as field length (maximum of 42)	Field's precision[1]
FT_INT	NUMBER	Same as field length (maximum of 42)	0
FT_LONG	NUMBER	Same as field length (maximum of 42)	0
FT_PACKED	NUMBER	Same as field length (maximum of 42)	Field's precision[1]
FT_SHORT	NUMBER	Same as field length (maximum of 42)	0
FT_UNSIGNED	NUMBER	Same as field length (maximum of 42)	0
FT_VARCHAR	LONG	Same as field length	
FT_ZONED	NUMBER	Same as field length (maximum of 42)	Field's precision

1. If the field length is greater than 42, the precision is adjusted using the calculation:
precision – field length + 42

To change these defaults you must edit the JPL procedure `type` in the distribution JPL module `oraf2t.jpl`, compile it by using `jpl2bin`, and replace the previous version in `orajpl.lib` by using `formlib -r`.

1.7.2
tbl2f

`tbl2f` creates a JAM form based on an ORACLE table. It creates a field for each column in the table, using the column's datatype to assign the appropriate field characteristics. The table below lists the default field lengths and precisions for each ORACLE datatype. It also lists a default JAM type.

Although ORACLE columns names are upper case, JAM/DB*i* by default uses lower case when creating field names for a `tbl2f` screen. This is consistent with the default case setting for ORACLE in `dbiinit.c` (see Section 1.1). If you changed the default in `dbiinit.c` to `DM_PRESERVE_CASE` or `DM_FORCE_TO_UPPER_CASE`, you should set the case option of `tbl2f` to match. The case option may be set on the command line or from a pull-down menu in interactive mode. For example, to start `tbl2f` in interactive mode and use upper case for JAM variables, type

```
tbl2f -i -lu
```

Note that there are additional characteristics associated with each JAM type. Those are described in the *Utility Reference Chapter* of the JAM/DB*i* manual.

ORACLE Type	JAM *Field Definition*		
	JAM *Type*	*Length*	*Precision*
NUMBER(0 length)	FT_FLOAT	16	5
NUMBER (0 precision)	FT_INT	Same as column length	
CHAR	FT_CHAR	Same as column length	
DATE	DT_DATETIME	20	
LONG	FT_VARCHAR	Same as column length (maximum of 255)	

To change these defaults, you must edit the JPL procedure `type` in the distribution JPL module `orat2f.jpl`, compile it by using `jpl2bin`, and replace the previous version in `orajpl.lib` by using `formlib -r`.

1.8 ORACLE-SPECIFIC COMMANDS

See Chapter 11

JAM/DB*i* for ORACLE provides additional commands for ORACLE-specific features. 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.

1.8.1 Using Transactions

JAM/DB*i* supports the following commands when using transactions. See the reference pages for more information on each command.

`AUTOCOMMIT`	turn on or off autocommit processing
`COMMIT`	commit a transaction
`ROLLBACK`	rollback a transaction

ORACLE transactions are per connection. An application must test for errors during the transaction and terminate a transaction by issuing an explicit `ROLLBACK` or `COMMIT`.

When an application closes a connection with `CLOSE_ALL_CONNECTIONS` or `CLOSE CONNECTION`, ORACLE commits any pending transactions on those connections. If an application terminates without explicitly closing its connections, ORACLE rolls back any pending transactions on those connections. However, these procedures are not recommended. Instead, it is strongly recommended that applications use explicit `COMMIT` and `ROLLBACK` statements to terminate transactions.

AUTOCOMMIT

turn autocommit on or off

SYNOPSIS

```
dbms [WITH CONNECTION connection] AUTOCOMMIT ON
dbms [WITH CONNECTION connection] AUTOCOMMIT OFF
```

DESCRIPTION

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

If the WITH CONNECTION clause is not used, JAM/DB*i* applies the AUTOCOMMIT setting to the application's default connection.

The default mode 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. Once a transaction is terminated, the engine automatically begins a new transaction.

Developers may change the default behavior by using the AUTOCOMMIT ON mode. In this mode, a statement is committed automatically upon successful execution. Its effects are immediately visible to other users, and it cannot be rolled back.

ORACLE recommends AUTOCOMMIT OFF mode because it may improve performance.

In AUTOCOMMIT OFF mode, an application should issue a COMMIT at the end of each logical unit of work. It should also use an error handler to test for errors and perform rollbacks as needed.

RELATED FUNCTIONS

```
dbms [WITH CONNECTION connection] COMMIT
dbms [WITH CONNECTION connection] ROLLBACK
```

EXAMPLE

```
proc tran_handle
   vars jpl_retcode
   retvar jpl_retcode
   dbms WITH CONNECTION xxx1 AUTOCOMMIT OFF
   jpl update_emp
# If the procedure "update_emp" executes successfully,
# jpl_retcode = 0; if a statement fails, jpl_retcode = -1
# or the value returned by the installed error handler.
# For all errors, execute a ROLLBACK.
   if jpl_retcode
   {
      dbms ROLLBACK
      msg emsg "New employee data NOT entered."
   }
   else
      msg emsg "New employee data successfully entered."
return 0

proc new_emp
   sql INSERT INTO emp \
   (ssn, last, first, street, city, st, zip, grade) VALUES
      (:+ssn, :+last, :+first, \
      :+street, :+city, :+st, :+zip, :+grade)
   sql INSERT INTO review (ssn, revdate, newsal, newgrd) \
      VALUES (:+ssn, :+hiredate, :+sal, :+grd)
   sql INSERT INTO acc (ssn, sal, exmp) \
      VALUES (:+ssn, :+sal, :+exmp)
   dbms COMMIT
   return 0
```

COMMIT

commit a transaction

SYNOPSIS

```
dbms [WITH CONNECTION connection] COMMIT
```

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 DBMS ROLLBACK, the updates are not committed, and the database is restored to its state prior to the start of the transaction. Once a transaction is terminated, the engine automatically begins a new transaction.

If the WITH CONNECTION clause is not used, JAM/DB*i* issues the commit on the default engine.

Before beginning a transaction, the application should ensure that the connection is using AUTOCOMMIT OFF mode; this is usually the default. It should COMMIT or ROLLBACK any pending transactions before starting a new one.

If an application is using AUTOCOMMIT ON mode, this command is not needed.

RELATED FUNCTIONS

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

EXAMPLE

Refer to the example shown for AUTOCOMMIT.

ROLLBACK

rollback a transaction

SYNOPSIS

dbms [WITH CONNECTION *connection*] ROLLBACK

DESCRIPTION

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

If the WITH CONNECTION clause is not used, JAM/DB*i* issues the rollback on the default engine.

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

RELATED FUNCTIONS

dbms [WITH CONNECTION *connection*] AUTOCOMMIT {ON | OFF}

dbms [WITH CONNECTION *connection*] COMMIT

EXAMPLE

Refer to the example shown for AUTOCOMMIT.

1.9
COMMAND DIRECTORY FOR ORACLE

This section contains a directory for all the commands available in JAM/DB*i* for ORACLE. The following table lists the command, a short description of the command, and the location of the reference page for that command. If the location is described as ORACLE Notes, that information is enclosed in this document.

Command	*Description*	*Documentation*
ALIAS	name a JAM variable as the destination of a selected column or aggregate function	JAM/DB*i* Manual
AUTOCOMMIT	turn on or off autocommit processing	ORACLE Notes
BINARY	create a JAM/DB*i* variable for fetching binary values	JAM/DB*i* Manual
CATQUERY	redirect SELECT results to a file or a JAM variable	JAM/DB*i* Manual
CLOSE CONNECTION	close a named connection	JAM/DB*i* Manual
CLOSE CURSOR	close a cursor	JAM/DB*i* Manual
CLOSE_ALL_CONNECTIONS	close all connections on all engines	JAM/DB*i* Manual
COMMIT	commit a transaction	ORACLE Notes
CONNECTION	set a default connection and engine for the application	JAM/DB*i* Manual
CONTINUE	fetch the next screenful of rows from a SELECT set	JAM/DB*i* Manual
CONTINUE_BOTTOM	fetch the last screenful of rows from a SELECT set	JAM/DB*i* Manual

Command	*Description*	*Documentation*
CONTINUE_DOWN	fetch the next screenful of rows from a SELECT set	JAM/DB*i* Manual
CONTINUE_UP	fetch the previous screenful of rows from a SELECT set	JAM/DB*i* Manual
CONTINUE_TOP	fetch the first screenful of rows from a SELECT set	JAM/DB*i* Manual
DECLARE CONNECTION	declare a named connection to an engine	JAM/DB*i* Manual
DECLARE CURSOR	declare a named cursor	JAM/DB*i* Manual
ENGINE	set the default engine for the application	JAM/DB*i* Manual
EXECUTE	execute a named cursor	JAM/DB*i* Manual
FORMAT	format the results of a CATQUERY	JAM/DB*i* Manual
OCCUR	set the number of rows for JAM/DB*i* to fetch to an array and choose an occurrence where JAM/DB*i* should begin writing result rows	JAM/DB*i* Manual
ONENTRY	install a JPL procedure or C function which JAM/DB*i* will call before executing a sql or dbms statement	JAM/DB*i* Manual
ONERROR	install a JPL procedure or C function which JAM/DB*i* will call whenever a sql or dbms statement fails	JAM/DB*i* Manual
ONEXIT	install a JPL procedure or C function which JAM/DB*i* will call after executing a sql or dbms statement	JAM/DB*i* Manual

Command	*Description*	*Documentation*
`ROLLBACK`	rollback a transaction	ORACLE Notes
`START`	set the first row for JAM/DB*i* to return from a `SELECT` set	JAM/DB*i* Manual
`STORE FILE`	store the rows of a `SELECT` set in a temporary file so that the application may scroll through the rows	JAM/DB*i* Manual
`UNIQUE`	suppress repeating values in a selected column	JAM/DB*i* Manual
`WITH CONNECTION`	set the default connection for the duration of a command	JAM/DB*i* Manual
`WITH CURSOR`	specify the cursor to use for a statement	JAM/DB*i* Manual
`WITH ENGINE`	set the default engine for the duration of a command	JAM/DB*i* Manual