Applixware Data
ELF Reference

This manual was produced using Applixware.

Printed: June 2010

DBASE_BUILD_SQL_SOURCE@

Creates an SQL query statement

Format query = DBASE_BUILD_SQL_SOURCE@()

Description Creates and returns the SQL query statement defined by the current query and table selections as an array of strings.

For example, query[0] contains the first string of the SQL query, query[1] contains the second string, and so on.

DBASE_COMMIT_EDITS@

Commits edits to the database

Format DBASE_COMMIT_EDITS@()

Description Commits changes made to the current table to the database. Once changes are committed, they are permanently entered into the database. The macro is for use with databases that have transactions.

See also DBASE_DISCARD_EDITS@

DBASE_CONDITIONS@

Sets the query conditions

Format DBASE_CONDITIONS@(format arrayof sql_condition_info@ conditions)

Arguments conditions An array of conditions.

Description Sets the query conditions of the current Data query to the passed array conditions. The structure sql_condition_info@, defined in the header file dbase_.am located in the */install_dir* /axdata/elf directory, contains the following information:

format sql_condition_info@

table1,	The table in the database containing column1
column1,	The comparison column for the condition
unused1,	(Not Used)
operator,	The comparison operator
not_it,	Preface operator with operator

value,	The comparison value for a Value query
valueList,	A list of values for "In" or "Between" query condition
table2,	The table in the database containing column2
column2,	The column used for a Column query condition
or_mode,	Boolean value, where TRUE means OR, FALSE means AND
unused2,	(Not Used)
sqlstr,	A subselect string for a Subselect query
type,	Type of query: "Value", "Column", or "Subselect"
front_parens,	The number of left parentheses before the condition
rear_parens,	The number of right parentheses after the condition

See also DBASE_GET_CONDITIONS@

DBASE_CONNECTION_RESET@

Resets server connection without resetting query conditions

Format DBASE_CONNECTION_RESET@(vendor, database, host, server, routing)

Arguments

vendor	The database vendor.
database	The name of the database.
host	The machine where the database resides.
server	The database server.
routing	The database connection routing.

Description Reestablishes a connection to a server without resetting query conditions. A connection is reestablished to the database. As the connection is reestablished, any edits you have made that have not been made are discarded. In addition, all cells in the table are cleared. Use DBASE_NEW_CONNECTION@ to reset a server connection and query conditions.

Errors can occur if you reuse conditions and establish a database connection to an environment that does not have the same features as the previous database connection.

The following are the supported database and the required arguments for each vendor.

Informix	database, machine.
Oracle	machine.
Sybase	dbase, server, machine.

See also DBASE_NEW_CONNECTION@

DBASE_COPY@

Copies the selected records to the clipboard

Format DBASE_COPY@()

Description Copies the selected records in the Data window to the clipboard. You can cut, copy, or paste between Data and Applixware Words or Applixware Spreadsheets.

See also DBASE_CUT@
DBASE_PASTE@

DBASE_CUT@

Removes selected records and places them in the clipboard

Format DBASE_CUT@()

Description Removes the selected Data records and places them in the clipboard. You can cut, copy, or paste between Data and Applixware Words or Applixware Spreadsheets.

See also DBASE_COPY@
DBASE_PASTE@

DBASE_DATABASE_LIST@

Returns a list of available databases

Format format sql_dbase_entry@ databaseList = DBASE_DATABASE_LIST@()

Description Returns a list of available databases for the current user. The structure sql_dbase_entry@, defined in the header file dbase_.am located in the /*install_dir* /axdata/elf directory, contains the following information:

format sql_dbase_entry@

vendor,	The database vendor
database,	The name of the database
host,	Machine running the ELF/SQL server
server,	The SQL server name on host
routing,	Vendor routing string
port,	NULL or port name

serviceName axnet service name to ELF/SQL server

DBASE_DELETE@

Deletes the selected records

Format DBASE_DELETE@()

Description Deletes the selected records from the table in the current Data window.

When the macro is used with databases that have transactions, the deletions are removed from the database when they are committed. When the macro is used with databases that do not have transactions, the deletions are instantly removed from the database.

DBASE_DELETE_FILE@

Deletes the current query

Format DBASE_DELETE_FILE@()

Description Deletes the current query and closes the Data window.

DBASE_DISCARD_EDITS@

Discards edits and closes the transaction

Format DBASE_DISCARD_EDITS@()

Description Discards additions, deletions, or changes made to the current table and rolls back the transaction to the database. The macro only discards edits made before the last **DBASE_COMMIT_EDITS@**. The macro can only be used with databases that have transactions.

DBASE_DO_QUERY_IN_FILE@

Loads a file and returns the query's result

Format format sql_result@ info = DBASE_DO_QUERY_IN_FILE@ (filename)

Arguments filename The name of an Applixware Data file

Description Loads an Applixware Data file containing a query and executes the query. The result of the query is returned as an sql_result@ formatted variable.

The definition of the sql_result@ format is as follows:

```
format sql_result@
                    'Column names and data types
    format col_info@ xxxyyy,
    rows            'The major data
```

The definition of the col_info@ format is as follows:

```
format col_info@
    names,          'Array of header names
    types           'Array of data types
```

DBASE_DOUBLE_CLICK_MACRO@

Names the macro invoked when a user double clicks in a cell

Format DBASE_DOUBLE_CLICK_MACRO@(name)

Arguments name The name of the macro that will be invoked.

Description Tells Applixware Data the name of the macro that will be invoked when a user double clicks in a cell. While only one macro can be active for any Applixware Data window, this macro does not need to be the same in each Applixware Data window.

DBASE_EXIT@

Closes Data and terminates the database server

Format DBASE_EXIT@()

Description Closes the current Data window, terminates the database server process, and closes the axnet connection.

DBASE_EXPORT_AS@

Exports the query results

Format DBASE_EXPORT_AS@(fileName[, format[, groupAccess[, allAccess]]])

Arguments fileName The name of the file being exported.

format The value representing the export file format. The valid file formats are:

1	Applixware Spreadsheets
2	WK1 (Lotus 1-2-3)
3	DIF
4	SYLK
5	CSV
6	ASCII GRID
7	ASCII ROW
8	ASCII COLUMN
9	WK3 (Lotus 1-2-3
10	XLS (Microsoft Excel 3.0)
11	XLS4 (Microsoft Excel 4.0)
12	Not Used
13	Applixware Words
14	RTF
15	XLS5 (Microsoft Excel 5.0)

groupAccess

The read and write file permissions for a member of the same group. The possible values are:

0	No read or write permission
1	Read-only
2	Read and write permission

allAccess The read and write file permissions for any user. The possible values are:

0	No read or write permission
1	Read-only
2	Read and write permission

Description Exports the information returned by a Data query to another file format. The exported information is saved as fileName and is given the passed file access privileges.

DBASE_FETCH_ALL@

Fetches all rows

Format DBASE_FETCH_ALL@()

Description The macro fetches all rows matching the current query conditions. The macro also re-queries the database, if necessary, to fetch all rows.

DBASE_FINISH_EDITING@

Ends transaction or edit

Format flag = DBASE_FINISH_EDITING@()

Description Either completes the current transaction or finishes the already started edit. If the database has transactions, the user is asked it the current changes should be committed or discarded.

A Boolean value is returned where FALSE indicates the user stopped the transaction from being completed.

DBASE_GET_ALL_STATE@

Returns an array of the Data file state

Format format **sql_state@** state = DBASE_GET_ALL_STATE@()

Description Returns the state of the current file attributes. The structure sql_state@, defined in the header file dbase_.am located in the /*install_dir* /axdata/elf directory, contains the following information:

format sql_state@

revstamp,	The revision number of the file, begins with 0 for a new file.
reserved1,	(Reserved)
save_mode,	DBASE#SAVE_QUERY_ONLY or DBASE#SAVE_WITH_DATA, defined in dbase.elh.
not_used1,	(Not Used)
not_used2,	(Not Used)
vendor,	Vendor name.
not_used3,	(Not Used)
not_used4,	(Not Used)
database,	The name of the database against which the query is made.
host,	Host computer name.
server,	The SQL server name on the machine.
routing,	The vendor routing string.
sqlstr,	An array of strings containing the actual SQL source code.
format arrayof sql_heading@ headers,	The column headers information.
format arrayof sql_table_info@ tables,	The tables involved in the query.

format sql_struct@ asql,
The sort, conditions, group by, and having, information.
not_used5, (Not Used)
not_used6, (Not Used)
not_used7, (Not Used)
interactive, Defines user interaction.
tableWinHeight, The window height of the Data window
tableWinWidth, The window width of the Data window
not_used8, (Not Used)
not_used9, (Not Used)
format sql_result@ savedData,
The saved results of a query, if save_mode = DBASE#SAVE_WITH_DATA
not_used10, (Not Used)
not_used11, (Not Used)
not_used12, (Not Used)
not_used13, (Not Used)
format sql_aliases_info@ alias
Alias information
format sql_char_attr_type@ font,
Font characteristics of Data window
update_validator,
The update validation macro name
not_used14, (Not Used)
double_click_macro
The macro called on a double-click in a row

DBASE_GET_AUTO_QUERY@

Returns the auto-query state

Format flag = DBASE_GET_AUTO_QUERY@()

Description Returns a Boolean value indicating the auto-query state. TRUE means query after each state change in the Data query. FALSE means query only on a manual query.

See also DBASE_SET_AUTO_QUERY@

DBASE_GET_BINARY_DATA@

Reads a blob of data

Format DBASE_GET_BINARY_DATA@(row, column)

Arguments

row	A row in the current table.
column	A column in the current table.

Description Displays a blob (binary large object) into a table's cell.

DBASE_GET_CHANNEL@

Returns RPC communications channel

Format channel = DBASE_GET_CHANNEL@()

Description Returns the RPC communications channel being used to communicate with a database.

DBASE_GET_COLUMNS@

Returns a list of columns from a table

Format nameArray = DBASE_GET_COLUMNS@(tablename)

Arguments tablename A database table.

Description Queries the database for the passed tablename and returns a two-dimensional array nameArray as follows:

nameArray[0]	Contains the column titles
nameArray[1]	Contains the corresponding column data type.

DBASE_GET_COLUMN_VALUES@

Returns an array of column values

Format valueArray = DBASE_GET_COLUMN_VALUES@ (tableName, columnName)

Arguments tableName A table within the current query and chosen database.

columnName
A column within the table.

Description Returns a unique array of all values in the named column.

DBASE_GET_CONDITIONS@

Returns the current query condition

Format format sql_condition_info@ conditions = DBASE_GET_CONDITIONS@()

Description Returns a structure containing the query condition that is part of the current Data query. A Data query is made up of one or more conditions. The structure sql_condition_info@, defined in the header file dbase_.am located in the /*install_dir* /axdata/elf directory, contains the following information:

format sql_condition_info@

table1,	The table in the database containing column1
column1,	The comparison column for the condition
unused1,	(Not Used)
operator,	The comparison operator; the comparison operators are:
	< Less than
	<= Less than or equal
	= Equal (basic operator)
	>= Greater than or equal
	> Greater than
	In In (basic operator)
	Between Between (basic operator)
	Like Like (basic operator)
	Is Null Is Null (basic operator)
not_it,	Preface basic operator with NOT
value,	The comparison value for a Value query.
valueList,	A list of values for "In" or "Between" query condition
table2,	The table in the database containing column2
column2,	The column used for a Column query condition.
or_mode,	Boolean value, where TRUE means OR, FALSE means AND
unused2,	(Not Used)
sqlstr,	A subselect string for a Subselect query
type,	Type of query: "Value", "Column", or "Subselect"
front_parens,	The number of left parentheses before the condition; the macro is used to determine the grouping of conditions in a query
rear_parens	The number of right parentheses after the condition; the macro is used to determine the grouping of conditions in a query

See also DBASE_CONDITIONS@

DBASE_GET_CONNECTION_INFO@

Returns database and connection information

Format format sql_dbase_entry@ entry = DBASE_GET_CONNECTION_INFO@()

Description Returns database, host, vendor, routing, and server information. The definition of sql_dbase_entry@ is as follows:

```
sql_dbase_entry@
        vendor,         'Oracle, Informix, ...
        database,       'Engine's name of database
        host,           'Machine running ELF/SQL server
        server,         'The SQL server name on the machine (Sybase)
        routine,        'Vendor routing string (for example, t:node:database in Oracle)
        port,           'Null or port name
        serviceName
                        'Axnet service name to ELF/SQL server
```

DBASE_GET_RECORDS@

Retrieves records from database

Format records = DBASE_GET_RECORDS@ (startRecord[,quantity])

Arguments

startRecord	The starting record, which is 0 based.
quantity	An optional value indicating the number of records to retrieve. If quantity is supplied, Data returns no more than the supplied quantity of records.

Description Retrieves an array of records from the database.

DBASE_GET_RESULTS@

Returns the results of a query

Format format sql_result@ info = DBASE_GET_RESULTS@(startingRecord [, count])

Arguments startingRecord
The record number; this is a zero-based value indicating the first record retrieved.

count The number of records to be returned. If this value is omitted, Applixware Data uses a default value.

Description Returns the results of a query.

DBASE_GET_SORT_INFO@

Returns sort information

Format format arrayof sql_sort_info@ sort = DBASE_GET_SORT_INFO@()

Description Returns an array of structures containing sort information for the current Data query. The structure sql_sort_info@, defined in the header file dbase_.am located in the /*install_dir* /axdata/elf directory, contains the following information:

format sql_sort_info@

column_name, The name of the column determining the sort

descending Boolean value where TRUE means descending sort, and FALSE means ascending sort

DBASE_GET_SOURCE@

Returns SQL source code

Format source = DBASE_GET_SOURCE@()

Description Returns an array of strings containing the SQL source code for the current Data query.

DBASE_GET_TABLES@

Returns table names

Format tableArray = DBASE_GET_TABLES@([username, [database}])

Arguments username User name.

database An array of database names (Sybase only)

Description Queries the database for the tables owned by username and returns an array of tables. If no username is passed, returns all tables in the current database. If database is passed, the macro searches for all databases in the array database.

See also DBASE_SET_TABLES@

DBASE_GET_TABLES_USED@

Returns query table information

Format format arrayof sql_table_info@ tables = DBASE_GET_TABLES_USED@()

Description Returns a structure containing information about tables used in the current Data query. The structure sql_table_info@, defined in the header file dbase_.am located in the /*install_dir* /axdata/elf directory, contains the following information:

format sql_table_info@

name,	The table name
uid,	User ID
colnames	The array of column names

DBASE_GET_TRANSACTION_USAGE@

Indicates if transactions are used

Format flag = DBASE_GET_TRANSACTION_USAGE@()

Description Returns a Boolean value indicating if transactions are used when the Data query interacts with a database. TRUE means use transactions with the database. FALSE means do not use transactions; instead actions are performed on individual records.

This macro indicates if transactions are used. In contrast, the macro DBASE_HAS_TRANSACTIONS@ indicates if transactions are available in the database.

DBASE_GET_TRIGGER@

Returns a trigger query's definition

Format trigger = DBASE_GET_TRIGGER@( )

Description Returns the value of the SQL select statement that is attached to the query as a trigger. For more information, see **DBASE_SET_TRIGGER@**.

DBASE_GET_UPDATE_VALIDATOR@

Returns update macro

Format macroName = DBASE_GET_UPDATE_VALIDATOR@()

Description Returns the name of the ELF macro that validates data before it is updated.

DBASE_GOTO_BEGINNING@

Scrolls to top

Format DBASE_GOTO_BEGINNING@()

Description Scrolls the table so that the first row of the retrieved information is made the current and displayed row.

See also **DBASE_GOTO_END@**

DBASE_GOTO_END@

Scrolls to bottom

Format DBASE_GOTO_END@()

Description Scrolls the table so that the last row of the information is made the current and displayed row. Because this command is displaying the last row, this macro may need to get all the data requested by the SELECT statement before it can move to this row.

See also **DBASE_GOTO_BEGINNING@**

DBASE_HAS_TRANSACTIONS@

Indicates if database transactions are available

Format flag = DBASE_HAS_TRANSACTIONS@()

Description Returns a Boolean value where TRUE means the database can have transactions. FALSE means the database does not have transactions, instead a record is updated when the cursor leaves the row being changed.

This macro indicates if a database can use transactions, while DBASE_GET_TRANSACTION_USAGE@ indicates if transactions are used.

DBASE_INSERT@

Inserts a row before the passed row

Format DBASE_INSERT@([beforeRow])

Arguments beforeRow The passed row.

Description Inserts a row in the current table. If beforeRow is included, a row is inserted before the current row. If the row number is NULL, a row is inserted after the current row.

DBASE_IS_ANSI@

Indicates if a database is an ANSI database

Format flag = DBASE_IS_ANSI@()

Description Returns a Boolean value where TRUE means the current database is a SQL database in ANSI SQL mode. FALSE means the database is in vendor mode.

DBASE_IS_INSET@

Returns a value indicating if Data is an inset

Format flag = DBASE_IS_INSET@()

Description Returns a Boolean value where TRUE means the current Data document is an inset in an Applixware Words or Spreadsheets document.

DBASE_LOAD_FILE@

Loads a Data file

Format DBASE_LOAD_FILE@(fileName)

Arguments fileName The passed Data file.

Description Loads the file containing query information and settings into the current Data window.

DBASE_MAIL@

Mails the current Data query

Format DBASE_MAIL@()

Description Mails the current Data query and retrieved information.

DBASE_MAX_RECORDS@

Sets the maximum number of records that can be retrieved

Format DBASE_MAX_RECORDS@(value)

Arguments value The passed quantity of records value.

Description Sets the maximum number of records that can be retrieved in one query operation.

DBASE_GET_DATA_FROM_FILE@

Transforms Applixware Data information into an ELF array

Format format table_data@ data = DBASE_GET_DATA_FROM_FILE@(datafile, mustDoQueryFlag)

Arguments datafile The name of a file containing Applixware Data information.

mustDoQueryFlag
A Boolean value which if set to TRUE indicates that the database will be requeried to create a new table.

Description Transforms the data stored within an Applixware Data file into an ELF array whose format is table_data@. This format is defined in the ELF include file table_.am.

The definition of this format is:

```
format table_data@
    data,           'an array of data, if data is TABLE_DATATYPE_ELF and element is an
                    array, then it is a row.
                    'about the data (may be NULL, or missing)
    format table_info@ info

format table_info@
                     'may be NULL
    format table_heading_info@   heading,
                    'may be NULL
    format arrayof table_column_info@ column

format table_heading_info@
    display         'TRUE if headers should be displayed

format table_column_info@
    name,           'string for column header
    width,          'in pixels, may be NULL
    type,           'see Data Types, below.
                    'for contents of cells, may be NULL
    format table_attributes@ attributes,
                    'for column headers, may be NULL
    format table_attributes@ heading_attributes

format table_attributes@  'any element may be NULL
    alignment,      'text alignment
    face,           'face name
    size,           'point size
    bold,           'true or false
    italic,         'true or false
    underline,      '0, 1 (single), or 2 (double)
    color           'as color is specified to the db_ctrl_text_color@() macro.
```

DBASE_GET_DATABASES_AVAIL@

Returns the name of available databases (Sybase only)

Format nameArray = DBASE_GET_DATABASES_AVAIL@( )

Description Returns the names of the databases from which data may be extracted. This macro can only be used with Sybase databases.

DBASE_GET_DATABASES_USED@

Returns list of databases that are currently being accessed (Sybase only)

Format list = DBASE_GET_DATABASES_USED@( )

DBASE_GET_DOC_INFO@

Returns a file name and attributes

Format format **doc_format** fileInfo = DBASE_GET_DOC_INFO@()

Description Returns a structure containing the file name and attributes of the current Data query.

DBASE_GET_DOUBLE_CLICK_MACRO@

Returns the name of macro invoked when a user double clicks in a cell

Format macroName = DBASE_GET_DOUBLE_CLICK_MACRO@( )

DBASE_GET_DUPLICATES_INFO@

Indicates if retrieved records can contain duplicates

Format flag = DBASE_GET_DUPLICATES_INFO@()

Description Returns a Boolean value that indicates if duplicate records are allowed in the retrieved data. TRUE means "allow no duplicates", FALSE means "allow duplicates".

DBASE_GET_EDIT_MODE@

Retrieves edit mode value

Format flag = DBASE_GET_EDIT_MODE@()

Description Retrieves a Boolean value indicating if edits can be made to the retrieved information or if new records can be added. TRUE means the query is in edit mode. FALSE means the query is not in edit mode.

See also DBASE_SET_EDIT_MODE@

DBASE_GET_EXPORTDOC@

Returns the attributes of an exported query

Format format doc_format_ exportInfo = DBASE_GET_EXPORTDOC@()

Description Retrieves a structure containing the attributes of a file exported from the current Data query.

See also DBASE_SET_EXPORTDOC@

DBASE_GET_FILE_INFO@

Returns information describing file header and permissions

Format format write_data_format@ file_info = DBASE_GET_FILE_INFO@()

Description Returns a data structure containing information describing the file's read/write permissions and information used within the Applixware file header.

The definition of write_data_format@ is as follows:

```
format write_data_format@
    comments,       ' Array of strings to be added to file as comments
    grp_access,     ' 0-none, 1-read, 2-write
    all_access,     ' 0-none, 1-read, 2-write
                    'specs for header line
    format afile_info file_header
```

The definition of afile_info is as follows:

```
format afile_info
    encoding,          ' TRUE if file is encoded
    version,           ' version number of current format
    docType,           ' as in recgfil_.am
    original_version, ' version number of original document
                       ' last version capable of reading this file
    minimum_version,
    content_hint       ' arbitrary hint string
```

DBASE_GET_FILENAME@

Returns file name of current query

Format fileName = DBASE_GET_FILENAME@()

Description Returns the file name of the current Data query.

See also DBASE_SET_FILENAME@

DBASE_GET_FONT@

Returns font for current query

Format format sql_char_attr_type@ font = DBASE_GET_FONT@()

Description Returns a structure containing the font characteristics for the current Data query. The structure sql_char_attr_type@ contains the following attributes:

```
format sql_char_attr_type@
    typeface,      The character typeface
    size,          The character point size
    bold,          Boolean value, where TRUE means bold characters
    italic         Boolean value, where TRUE means italicize characters
```

See also DBASE_SET_FONT@

DBASE_GET_GROUP_BY_INFO@

Retrieves "group by" columns

Format groupByColumnsArray = DBASE_GET_GROUP_BY_INFO@()

Description Returns the "group by" columns in the current Data query.

See also DBASE_SET_GROUP_BY_INFO@

DBASE_GET_HAVING_INFO@

Returns "having" Data conditions

Format format sql_condition_info@ conditions = DBASE_GET_HAVING_INFO@()

Description Returns a structure containing the query "having" conditions for the current Data query. A "having" Data query is made up of one or more "having" conditions. The structure sql_condition_info@, defined in the header file dbase_.am located in the /*install_dir* /axdata/elf directory, contains the following information:

format sql_condition_info@

table1,	The table in the database containing column1
column1,	The comparison column for the condition
unused1,	(Not Used)
operator,	The comparison operator; the operators that you can specify are:
	< Less than
	<= Less than or equal
	= Equal (basic operator)
	>= Greater than or equal
	> Greater than
	In In (basic operator)
	Between Between (basic operator)
	Like Like (basic operator)
	Is Null Is Null (basic operator)
not_it,	Preface operator with NOT
value,	The comparison value for a Value query
valueList,	A list of values for an "In" or "Between" query condition
table2,	The table in the database containing column2
column2,	The column used for a Column query condition
or_mode,	Boolean value where TRUE means OR and FALSE means AND
unused2,	(Not Used)
sqlstr,	A subselect string for a subselect query
type,	Type of query: "Value", "Column", or "Subselect"
front_parens,	The number of left parentheses before the condition; the macro is used to determine the grouping of conditions in a query
rear_parens	The number of right parentheses after the condition; the macro is used to determine the group of conditions in a query

See also DBASE_SET_HAVING_INFO@

DBASE_GET_HEADERS@

Returns column heading attributes

Format format sql_heading@ headers = DBASE_GET_HEADERS@()

Description Returns a structure of column heading attributes in the current Data query. The structure sql_heading@, defined in the header file dbase_.am located in the /*install_dir* /axdata/elf directory, contains the following attributes:

format sql_heading@

name,	The column name to use in the display
width,	The column width in dots per inch (DPI)
type,	The type of column data
db_name,	The database, table, and column name for the column, in database.table.colName format
alignment	The value for alignment: 1 Left 2 Center 3 Right

See also DBASE_SET_HEADERS@

DBASE_GET_INTERACTIVE@

Indicates if SQL source is edited

Format flag = DBASE_GET_INTERACTIVE@()

Description Returns a Boolean value where TRUE means the current Data query SQL source is the original source. FALSE means the SQL source was modified in the Edit SQL Source dialog box in Data.

See also DBASE_SET_INTERACTIVE@

DBASE_GET_JOIN_INFO@

Returns join information

Format format arrayof sql_join_info@ join = DBASE_GET_JOIN_INFO@()

Description Returns an array of structures containing join information for the current Data query. The structure sql_join_info@, defined in the header file dbase_.am located in the /*install_dir* /axdata/elf directory, contains the following attributes:

format sql_join_info@

table1,	Name of the first table
table2,	Name of the second table
column1,	Name of the join column in table1
column2,	Name of the join column in table2
outer1,	Boolean value where TRUE means a right outer join
outer2,	Boolean value where TRUE means a left outer join
operator,	One of the following comparison operators:

=	equal
!=	not equal
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal

See also DBASE_SET_TABLES@

DBASE_GET_MAX_RECORDS@

Indicates the maximum number of records that can be retrieved

Format value = DBASE_GET_MAX_RECORDS@()

Description Returns a value indicating the maximum number of records that can be retrieved.

DBASE_GET_MODIFIED@

Indicates if records are edited

Format flag = DBASE_GET_MODIFIED@()

Description Returns a Boolean value indicating if records are edited. TRUE means the retrieved records within a table were edited. FALSE means the retrieved records are unchanged.

DBASE_NEW@

Starts a new Data query

Format DBASE_NEW@(vendor, database, host, server, routing)

Arguments

vendor	The database vendor.
database	The name of the database.
host	The machine where the database resides.
server	The database server.
routing	The database connection routing.

Description Starts a new Data query in the current Data window. A connection is established to the passed database. The following are the supported database vendors and the required arguments for each vendor.

Informix	database, host.
Oracle	host.
Sybase	dbase, server, host.

DBASE_NEW_CONNECTION@

Resets server connection without disturbing fonts, filename, etc.

Format DBASE_NEW_CONNECTION@(vendor, database, host, server, routing)

Arguments

vendor	The database vendor.
database	The name of the database.
host	The machine where the database resides.
server	The database server.
routing	The database connection routing.

Description Reestablishes a connection to a server and starts a new query. A connection is reestablished to the database. As the connection is reestablished, any edits you have made that have not been made are discarded. In addition, all cells in the table are cleared.

The following are the supported database and the required arguments for each vendor.

Informix database, machine.

Oracle machine.

Sybase dbase, server, machine.

See also DBASE_NEW@

DBASE_OPEN_FROM_WP@

Opens a an Applixware Data window from within Applixware Words

Format DBASE_OPEN_FROM_WP@(nameArray)

Arguments nameArray An array containing the following information:

nameArray[0]	Document name
nameArray[1]	Window title

If you omit the nameArray[1] windowTitle element, you can use a string as an argument to this macro

Description Opens an Applixware Data window that is inset within an Applixware Words document. If the *autoquery* preference is set, the database will be queried and the data within the window will be updated.

DBASE_OVERRIDE_TITLE@

Changes Data window title

Format DBASE_OVERRIDE_TITLE@(title)

Arguments title The new title.

Description Changes the title of the current Data window to the passed string.

DBASE_PASTE@

Inserts information in the Data window

Format DBASE_PASTE@([row], [format table_data@ data])

Arguments row A zero-based number indicating the row to insert the information. If row is not supplied, the information is inserted above the currently edited row.

data An ELF table_data@ format. The table_data@ format is defined in the file table_.am.

Description Inserts information as a record in the current Data window. The macro pastes the information before the passed row.

Both arguments are optional.

See also DBASE_COPY@

DBASE_CUT@

TABLE_.AM

DBASE_QUERY_FROM_FILE@

Returns records from a query file

Format records = DBASE_QUERY_FROM_FILE@(fileName)

Arguments fileName The passed file name.

Description Returns an array of records from the Data query fileName.

DBASE_RECORD_MACRO@

Records a macro

Format DBASE_RECORD_MACRO@()

Description Records a macro within the current Data window.

DBASE_REQUERY@

Requeries the database

Format DBASE_REQUERY@()

Description Requeries the database using current Data query statement.

DBASE_RESIZE@

Resizes the current Data window

Format DBASE_RESIZE@(width, height)

Arguments

width	The new window width.
height	The new window height.

Description Resizes the current Data window to width and height.

DBASE_REVERT@

Restores a Data file to its previous saved state

Format DBASE_REVERT@()

Description Restores a Data file to its previous saved state. The Data window remains open and all uncommitted edits are discarded. Unlike File ® Revert, DBASE_REVERT@ does not prompt to verify that the user really wants to restore the document and discard any edits.

DBASE_RUN_SQL@

Runs the passed SQL file

Format DBASE_RUN_SQL@(sqlSource)

Arguments

sqlSource	An SQL source file.

Description Runs the passed SQL source file. The file is an array of strings containing SQL code.

DBASE_SAVE_AS@

Saves the current file to a new name

Format DBASE_SAVE_AS@(name[, saveMode[, groupAccess[, allAccess]]])

Arguments

name	The name of the new file.

saveMode The save mode of the query. The possible values are:

0 Save the query without saving current data
1 Save the query with all data matching query conditions, fetching all rows.
2 Save the query with current data

groupAccess The read and write file permissions for a member of the same group. The possible values are:

0 No read or write permission
1 Read-only
2 Read and write permission

allAccess The read and write permissions for any user. The possible values are:

0 No read or write permission
1 Read-only
2 Read and write permission

Description Renames the current file to name and saves it to disk. The file's access privileges are set using groupAccess and allAccess. The file's saveMode determines whether current data is saved with the query. When a query saved with current data is opened, the data represents the results of the last query, not the present state of the source table, until a query of the source table is initiated.

DBASE_SELECT_ALL@

Selects all records retrieved by the query

Format DBASE_SELECT_ALL@()

Description Selects all the records retrieved by the Data query .

DBASE_SET_AUTO_QUERY@

Sets the auto-query state

Format DBASE_SET_AUTO_QUERY@(flag)

Arguments flag A Boolean value.

Description Sets the auto-query state for the current Data query. TRUE means turn on auto-query for a query after each state change in the Data query. FALSE means turn off auto-query for a query only on a manual query.

See also DBASE_GET_AUTO_QUERY@

DBASE_SET_DATABASES_USED@

Names the databases from which tables can be chosen

Format DBASE_SET_DATABASES_USED@(list)

Arguments list An array of database names.

Description Defines the names of all databases from which data may be queried. That is, only tables from these tables may be queried.

See also DBASE_GET_DATABASES_USED@

DBASE_SET_DISPLAYED_COLUMNS@

Specifies which columns are displayed

Format DBASE_SET_DISPLAYED_COLUMNS@(columns)

Arguments columns An array of column names.

Description Indicates which of the query columns are displayed. A column can be used in the query without being displayed.

DBASE_SET_EDIT_MODE@

Sets the edit mode

Format DBASE_SET_EDIT_MODE@(flag)

Arguments flag A Boolean value indicating the edit mode.

Description Sets the edit mode for the current Data query. TRUE means allow edits. FALSE means do not allow edits.

See also DBASE_GET_EDIT_MODE@

DBASE_SET_EXPORTDOC@

Sets the export document information

Format DBASE_SET_EXPORTDOC@(format doc_format_ information)

Arguments information A structure containing the export document's information.

Description Sets the export document information for the file in the current Data window. The macro is used with **DBASE_EXPORT_AS@** to set the filename, group access, and all access export attributes of the exported document.

See also **DBASE_GET_EXPORTDOC@**

DBASE_SET_EXPRESSIONS@

Sets columns to the passed expressions

Format DBASE_SET_EXPRESSIONS@(expressions)

Arguments expressions The passed array of expression columns.

Description Sets the columns in the current Data query to the passed array of expressions. An expression is a mathematical function or an aggregate SQL function, such as sum (sum) or average (avg).

A simple mathematical expression is the addition, subtraction, multiplication, and so on, on two or more columns. For example, you can create a new column, column1+column2, that adds a record in column1 to the corresponding record in column2.

Aggregate functions provide summary values for column values. Aggregate functions are primarily used with group by and having queries. For example, avg(column1) finds the average of the groups of column1 values.

DBASE_SET_FILENAME@

Sets a Data query file name

Format DBASE_SET_FILENAME@(fileName)

Arguments fileName A file name.

Description Sets the name to which the current Data query will be saved to fileName.

See also DBASE_GET_FILENAME@

DBASE_SET_FONT@

Sets the font

Format DBASE_SET_FONT@(format sql_char_attr_type@ newFont)

Arguments newFont The passed font characteristics.

Description Sets the font in the current Data window to the passed font characteristics. The structure sql_char_attr_type@, defined in the header file dbase_.am located in the */install_dir* /axdata/elf directory, contains the following attributes:

format sql_char_attr_type@

typeface,	The character typeface
size,	The character point size
bold,	Boolean value, where TRUE means bold characters
italic	Boolean value, where TRUE means italic characters

See also DBASE_GET_FONT@

DBASE_SET_GROUP_BY_INFO@

Sets the "group by" columns

Format DBASE_SET_GROUP_BY_INFO@(columns)

Arguments columns The passed array of columns.

Description Sets the "group by" columns for the query to columns.

See also DBASE_GET_GROUP_BY_INFO@

DBASE_SET_HAVING_INFO@

Sets the "having" conditions

Format DBASE_SET_HAVING_INFO@(format arrayof sql_condition_info@ having)

Arguments having An array of structures containing "having" conditions.

Description Sets the query "having" conditions of the current Data query to the passed array of structures containing having conditions. The structure sql_condition_info@, , defined in the header file dbase_.am located in the /*install_dir* /axdata/elf directory, contains the following information:

format sql_condition_info@

table1,	The table in the database containing column1
column1	The comparison column for the condition.
unused1	(Not Used)
operator	The comparison operator. The valid comparison operators are:
	< Less than
	<= Less than or equal
	= Equal (basic operator)
	>= Greater than or equal
	> Greater than
	In In (basic operator)
	Between Between (basic operator)
	Like Like (basic operator)
	Is Null Is Null (basic operator)
not_it	Preface operator with operator
value	The comparison value for a Value query
valueList	A list of values for "In" or "Between" query conditions
table2	The table in the database containing column2
column2	The column used for a Column query condition
or_mode	Boolean value where TRUE means OR, FALSE means AND
unused2	(Not Used)
sqlstr	A subselect string for a Subselect query
type	Type of query: "Value", "Column", or "Subselect"
front_parens	The number of left parentheses before the condition; the macro determines the grouping of conditions in a query
rear_parens	The number of right parentheses after the condition; the macro determines the grouping of conditions in a query

See also DBASE_GET_HAVING_INFO@

DBASE_SET_HEADERS@

Sets the column heading attributes

Format DBASE_SET_HEADERS@(format sql_heading@ headers)

Arguments headers A structure of column heading attributes.

Description Sets the column heading attributes of the current data query to the passed structure of heading attributes.

See also DBASE_GET_HEADERS@

DBASE_SET_INTERACTIVE@

Sets the edited SQL source value

Format DBASE_SET_INTERACTIVE@(flag)

Arguments flag A Boolean value.

Description Sets the edited SQL source flag. TRUE means that the current Data query SQL source is the original source created by Data. FALSE means the SQL source was modified.

See also DBASE_GET_INTERACTIVE@

DBASE_SET_SORT_INFO@

Sets the source information

Format DBASE_SET_SORT_INFO@(format arrayof sql_sort_info@ sort)

Arguments sort The sort information.

Description Sets the sort information for the current data query. The structure sql_sort_info@, defined in the header file dbase_.am located in the /*install_dir* /axdata/elf directory, contains the following information:

format sql_sort_info@

column_name,	The name of the column determining the sort
descending	Boolean value where TRUE means descending sort, and FALSE means ascending sort

DBASE_SET_TABLES@

Sets the table and join information

Format DBASE_SET_TABLES@(format arrayof sql_table_info@ tables, format arrayof sql_join_info@ join)

Arguments

tables	An array of structures containing table information.
join	An array of structures containing join information.

Description Sets the table and join information for the current Data query. The structure sql_table_info@, defined in the header file dbase_.am located in the /*install_dir* /axdata/elf directory, contains the following information:

format sql_table_info@

name,	The table name
uid,	User ID
colnames	The array of column names

The structure sql_join_info@, defined in the header file dbase_.am located in the /*install_dir* /axdata/elf directory, contains the following attributes:

format sql_join_info@

table1,	Name of the first table
table2,	Name of the second table; if there is no join, table2 must be the same as table1
column1,	Name of the join column in table1
column2,	Name of the join column in table2.
outer1,	Boolean value where TRUE means a right outer join
outer2,	Boolean value where TRUE means a left outer join
operator	One of the following comparison operators:

=	equal
!=	not equal
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal

See also **DBASE_GET_JOIN_INFO@**
DBASE_GET_TABLES@

DBASE_SET_TRIGGER@

Sets the SQL Select statement that will act as a trigger

Format DBASE_SET_TRIGGER@(stringOrArray)

Arguments stringOrArray The select statement that will be the trigger.

Description Adds an SQL select statement to an Applixware Data document. This select statement is a query used in conjunction with the document's regularly defined query. You would use a trigger query to indicate that a change has occurred and, if the change meets your conditions, the database should be requeried to provide new information.

See also DBASE_GET_TRIGGER@

DBASE_SET_USER_ID@

Sets user name and password

Format DBASE_SET_USER_ID@(username, pwd)

Arguments

username	The user name.
pwd	The user password.

Description Sets the user name and password for a SYBASE or Oracle database session. Use the macro DBASE_USER_ID@ to retrieve this user name and password.

If no user name and password has been previously established, the macro DBASE_NEW_CONNECTION@ automatically calls DBASE_SET_USER_ID@ before attempting to connect to the database.

DBASE_SOURCE@

Changes the SQL source code

Format DBASE_SOURCE@(source)

Arguments

source	An SQL source code.

Description Changes the SQL source code of the current Data query to source. The current database is queried with the new source code. source is a simple string.

DBASE_SQL_NO_DUPLICATES@

Sets the "allow duplicates" status

Format DBASE_SQL_NO_DUPLICATES@(dupFlag)

Arguments dupFlag A Boolean value.

Description Sets the "allow duplicates" flag for the current Data query. TRUE means "allow no duplicates", FALSE means "allow duplicates".

DBASE_STATUS_MESSAGE@

Displays a status message

Format DBASE_STATUS_MESSAGE@(message)

Arguments message The status message string.

Description Displays a message in the status area of the current Data window.

DBASE_TABLE_EDITED@

Indicates if a table is edited

Format flag = DBASE_TABLE_EDITED@()

Description Returns a Boolean value where TRUE means the table rows have changed, FALSE means there are no changes.

DBASE_UPDATE_BINARY_DATA@

Changes a blob (binary large object)

Format DBASE_UPDATE_BINARY_DATA@(row, col, blob)

Arguments

row	The row containing the data being changed.
col	The column containing the data being changed.
blob	The data being inserted into the table.

Description Changes the *blob* (Binary Large Object) that is stored at (row, col) in the current table. If the value of blob is NULL, the database value is also set to NULL.

DBASE_UPDATE_VALIDATOR@

Sets the validation macro name

Format DBASE_UPDATE_VALIDATOR@(macroName)

Arguments macroName The validation macro name.

Description Sets the update validation macro name to the passed macroName. A validation macro checks column names and values in columns for updated and inserted rows.

DBASE_USER_ID@

Returns the current user name and password

Format information = DBASE_USER_ID@()

Description Returns an array containing the user name and password of the current Data query's user.

See also DBASE_SET_USER_ID@

dbase_.am

```
'
'
'
' Profiles:
'
define DBASE#PROF#JOIN_MACRO    "dbaseJoinMacro"
define DBASE#PROF#USER          "dbaseDefaultUser"
define DBASE#PROF#PASSWD        "dbaseDefaultPasswd"
define DBASE#PROF#DBASE         "dbaseDefaultDatabase"
define DBASE#PROF#PORT          "dbaseDefaultElfPort"
define DBASE#PROF#MACHINE       "dbaseDefaultHost"
define DBASE#PROF#MAX_RECORDS
                                "dbaseDefaultMaxRecords"
define DBASE#PROF#VENDOR        "dbaseDefaultVendor"
define DBASE#PROF#SERVER        "dbaseDefaultServer"
define DBASE#PROF#ROUTING       "dbaseDefaultRouting"
define DBASE#PROF#AUTO_QUERY
                                "dbaseAutoQuery"
define DBASE#PROF#TABLE_OWNER
                                "dbaseDefaultTableOwner"
define DBASE#PROF#AUTO_CONNECT
                                "dbaseAutoConnect"
define DBASE#PROF#FONT           "dbaseDefaultFont"
define DBASE#PROF#FONT_SIZE     "dbaseDefaultFontSize"
define DBASE#PROF#FONT_BOLD     "dbaseDefaultFontBold"
define DBASE#PROF#FONT_ITALIC   "dbaseDefaultFontItalic"
define DBASE#PROF#TABLE_LIST    "dbaseTableListingMacro"
define DBASE#PROF#FETCH_COUNT
                                "dbaseFetchRecordsCount"
define DBASE#PROF#DATETIME_FORMAT "dbaseDatetimeFormat"
define DBASE#PROF#TIMEOUT       "dbaseTimeout"
'
'
' Special Files:
'
define DBASE#DBASE_LIST "rdblist"    ' list of known databases in user's axhome
'
'
'
' Command codes used as the first arg to the update request macro
```

```
' specified by the dbase_update_validator@ function.  Note that
' when the funtion is called, a rejection should be in the form
' of a thrown error, not an info_message@ or return value.  The
' return value of the validator MUST be null.
'
define DBASE#UPDATE#RQST 0
define DBASE#DELETE#RQST 1
define DBASE#INSERT#RQST 2
'
'
'
' Poke Codes emitted by data dipper to subordinate dialogs.
'
define DBASE#CONNECTING          11001 ' New connection being made
define DBASE#CONNECTED           11002 ' New connection established
define DBASE#TABLES              11003 ' Tables list has changed
define DBASE#SELECTIONS          11004 ' Change of "highlights"
define DBASE#REQUERY             11005 ' Database queried for new data
define DBASE#FETCHING            11006 ' Getting next set of records
define DBASE#DBL_CLICKED         11007 ' Rows was double clicked
define DBASE#KEY_PRESS           11008 ' Key press
define DBASE#COLUMNS             11009 ' Columns displayed has changed
define DBASE#KEY_COLUMNS         11010 ' Key columns has changed
define DBASE#CREATED_COLUMNS     11011 ' New columns created
define DBASE#SORT                11012 ' Sort order changed
define DBASE#CONDITIONS          11013 ' Conditions changed
define DBASE#GROUP_BY            11014 ' Group by list changed
define DBASE#HAVING              11015 ' Having clause changed
define DBASE#HEADINGS            11016 ' Column headings changed
define DBASE#INTERACTIVE         11017 ' Interactive mode
define DBASE#FONT                11018 ' Font changed
define DBASE#COMMIT              11019 ' Commit edits
define DBASE#DISCARD             11020 ' Discard edits
'
'
' Data types to be specified for Sybase remote procedure execution
'
'
define SYB#CHAR_                 0x2f
define SYB#VARCHAR_              0x27
define SYB#TEXT_                 0x23
define SYB#BINARY_               0x2d
define SYB#VARBINARY_            0x25
define SYB#IMAGE_                0x22
```

```
define SYB#INT1_                0x30
define SYB#INT2_                0x34
define SYB#INT4_                0x38
define SYB#FLT8_                0x3e
define SYB#REAL_                0x3b
define SYB#BIT_                 0x32
define SYB#MONEY_               0x3c
define SYB#MONEY4_              0x7a
define SYB#DATETIME_            0x3d
define SYB#DATETIME4_           0x3a
'
'
' A Data Dipper Document is fundametally an Elf data file that contains
' a sql_state@ formated array:
'
format sql_state@               ' This is what's saved to disk
        revstamp,               ' Currently zero
        asc_header,             ' This is used only by Applixware Source Control system
        save_mode,              ' DBASE#SAVE_QUERY_ONLY,
                                DBASE#SAVE_WITH_DATA ...
        not_used1,
        not_used2,
        vendor,                 ' Oracle, Informix...
        not_used3,
        not_used4,
        database,               ' name of default database
        host,                   ' Host computer name
        server,                 ' the SQL server name on the machine (Sybase)
        routing,                ' vendor routing string (e.g. t:node:database in Oracle)
        sqlstr,                 ' array of strs that is the actual SQL code
        ' column headers name mapping
        format arrayof sql_heading@ headers,
        format arrayof sql_table_info@ tables,    ' tables involved in query
        format sql_struct@ asql,        ' the result of user's work
        database_list,          ' sybase only - all databases to query on
        not_used6,
        not_used7,
        interactive,            ' define user interaction (SQL source/interactive)
        tableWinHeight,
        tableWinWidth,
        not_used8,
        not_used9,
        format sql_result@ savedData, 'data, if save_mode = DBASE#SAVE_WITH_DATA
        not_used10,
```

```
        not_used11,
        not_used12,
        not_used13,
        format sql_aliases_info@ alias,    ' the alias information
        format sql_char_attr_type@ font,  ' font for whole of Data Dipper
        update_validator,
        triggerstr,                        ' used by rtsql for watch query
        double_click_macro                 ' called on row double-click

format sql_heading@
        name,                              ' Name to use in table
        width,                             ' in 75dpi's
        type,                              ' text, date...
        db_name,                           ' database table.colname
        alignment,                         ' 1 = Left, 2 = Center 3 = Right
        displayFormat                      ' display format. for datetime is datetime format. If null,
                                           ' use default.

format sql_dbase_entry@                    ' as returned from dbase_database_list@()
        vendor,                            ' Oracle, Informix...
        database,                          ' Engine's name of database
        host,                              ' Machine running elf/sql server
        server,                            ' the SQL server name on the machine (Sybase)
        routing,                           ' vendor routing string (e.g. t:node:database in Oracle)
        port,                              ' Null or port name
        serviceName                        ' Axnet service name to elf/sql server

format sql_table_info@
        name,
        uid,                               ' to be able to get columns stuff
        colnames                           ' if null, use uid to get columns

format sql_join_info@
        table1,                            ' name of table (not struct)
        table2,
        column1,                           ' name of column in table
        column2,
        outer1,                            '  outer join?
        outer2,                            ' for both tables (ANSI SQL)
        operate                            ' =,!=,<,<=,>,>=


format sql_condition_info@
        table1,
```

```
        column1,
        compare,
        operate,                        ' =, >, < ... '
        not_it,                         ' preface basic operator with NOT
        value,
        valueList,                      ' for between's and in's
        table2,
        column2,
        or_mode,                        ' if true, OR rather than and
        grp_lvl,                        ' OBSOLETE
        sqlstr,                         ' a subselect string
        type,                           ' value/column/subselect ?
        front_parens,                   ' no of  (left ) parens before
        rear_parens                     ' no of (right) parens  after the condition

format sql_sort_info@
        column_name,
        descending

format sql_struct@
        query_cols,                     ' if null, "*". only those in the select list
        unused,                         ' used to be tab_cols, but was never used
        expressions,                    ' all the computed columns
        format arrayof sql_join_info@ join,          ' list of tables to join
        format arrayof sql_condition_info@ conditions,
        group_cols,                     ' columns to "group by "
        format arrayof sql_condition_info@ havings,      ' columns may be expressions
        format arrayof sql_sort_info@ sort,
        no_duplicates

format sql_result@                      ' result of a query
        format col_info@ xxxyyy,        ' col names and data types
        rows                            ' the major data

format sql_aliases_info@
        db_names,                       '  actual names of the columns in the DBMS (array)
        pseudonyms,                     '  Name to use in display table (array)
        widths,                         '  in 75dpi's (array)
        types,                          '  type of data text, date... (array)
        alignments,   ' 1 = Left, 2 = Center 3 = Right (array)
        displayFormats

format sql_cursor_properties@
        fetch_size,                     ' num of records per fetch [50]
```

```
        trim_strings,                   ' trim strings? [SQL_TRIM_BOTH_]
        transpose_data,                 ' transpose data? [FALSE]
        nums_as_nums,                   ' numbers as numbers? [FALSE] (i.e. as strings)
        binary_size_limit, '(added 3.2)max size of binary data item fetched
        timeout, '(added 3.2)timeout in seconds, zero for infinite, null for default
        max_rows,    'maximum number of rows (passed to DBMS server), null for default
        dates_as_dates      'Dates as Dates? [FALSE] (i.e, as strings)

format col_info@                        ' headers info on sql_select@ when "headingsToo"
        names,                          ' array of heading names
        types                           ' array of datatypes, see sqltypes.h

'
' General Purpose defines
'
define DBASE#SAVE_QUERY_ONLY   0        ' Save Mode
define DBASE#SAVE_WITH_DATA    1        ' Save Mode

'
' Sybase stored procedures return a sql_procedure_result@ which
' is an array with three elements.
' The first element is an array composed of the results of the select
' statements executed by the stored procedure.  If there are two select
' statements, this is an array of 2 items. Each of the 2 contains a 2-d array
' which is the result of the select,
'
format sql_procedure_result@
        select_results,                 ' 3-d array of select results (one 2-d array per select)
        return_parameters,              ' return parameters from the stored procedure
        status                          ' status number returned by the procedure

format sql_procedure_params@
        data,                           ' data to be passed to the stored procedure
        type,                           ' type of data; SYB#INT1_, SYB#REAL_ etc
        output                          ' is it a return parameter?

'returned by sql_get_col@
format sql_collist@
        id,                             '"" for all or for expressions, otherwise table name
        sqlready,                        'fully qualified colnames and expressions
        userready,                       'semiqualified colnames w/aliases
        reserved

'information about external editors that are running to edit binary data items
```

```
format sql_editor@
        tempFile,                       'name of temp file that contains binary info
        taskId,                         'id of task launched to edit/view.
        docType,                        'type of document
        dateLastModified,               'when the tempFile was created from dbms data
        SQLTable,                       'table name
        SQLColumn,                      'column name
        SQLCondition,                   '"where" conditions for row. If null, insert.
        dbTableRow                      'db_table row number for this data

format sql_char_attr_type@
        face,
        size,
        bold,
        italic


format sql_channel@
        id,
        host

/* used by search and replace code */
format sql_search_args@
        find_str,                       'string to search for
        replace_str,                    'string to replace with
        colnames,                       'columns to search within
        row,                            'last previously found location
        col,                            'last previously found location
        forward,                        'true if search forward, false if search backward
        case_sensitive,                 'case sensitive search
        preserve_case,                  'case sensitive replace
        whole_entry,                    'match entire entry or substring
        match_found,                    'set by searcher to indicate whether match found
        format sql_result@ data         'the data set to search

/*
Interface between ELF & C for Elf database access.
SQL Gateway defines.

*/


#define ELFSQL_PING_            0
#define ELFSQL_CONNECT_         1
```

```
#define ELFSQL_DISCONNECT_      2
#define ELFSQL_LOGGING_         3
#define ELFSQL_PREPARE_         4

#define ELFSQL_UNPREPARE_       5
#define ELFSQL_DECL_CURSOR_     6
#define ELFSQL_OPEN_CURSOR_     7
#define ELFSQL_FETCH_CURSOR_    8
#define ELFSQL_CLOSE_CURSOR_    9

#define ELFSQL_EXEC_STR_        10
#define ELFSQL_EXEC_STMT_       11
#define ELFSQL_PREPARE_UPDT_    12
#define ELFSQL_SEND_BIN_        13
#define ELFSQL_CURSOR_PROPS_    14

#define ELFSQL_PROCEDURE_       15
#define ELFSQL_QUERY_FORMAT_    16      /* for non-SQL gateways */
#define ELFSQL_COL_NAMES_       17      /* for non-SQL gateways */
#define ELFSQL_TABLE_NAMES_     18      /* for non-SQL gateways */
#define ELFSQL_ACCEPT_TABLE_    19      /* for non-SQL gateways */
#define ELFSQL_DESCRIBE_        20
#define ELFSQL_PREPARE_WATCH_   21
#define ELFSQL_WATCH_QUERY_     22
#define ELFSQL_INTERRUPT_RECVD_         23
#define ELFSQL_PAUSE_           24 /* gateway will pause(), so debugger can attach */
#define ODBC_DATA_SOURCES_      25 /* for ODBC, get list of possible connections */
#define ODBC_TABLES_            26 /* get table names */
#define ODBC_COLUMNS_           27 /* get column names */
#define ODBC_DATASOURCES_       28 /* get list of data sources */
#define ODBC_PRIMARYKEYS_       29
#define ODBC_TRANSACT_          30
#define ODBC_GETINFO_           31
#define ODBC_SETPOS_            32

/*
 * The basic data types in RDBMS:
 */

#define SQL_CHAR_               1
#define SQL_NUMBER_             2
#define SQL_BINARY_             3

/*
```

```
 * The alignment information for the displayed columns. left, center
 * and right.
 */

#define SQL_ALIGN_L_            1
#define SQL_ALIGN_C_            2
#define SQL_ALIGN_R_            3

/*
 * Whether and where to trim the strings
 */

#define SQL_TRIM_BOTH_          0
#define SQL_TRIM_BEGIN_         1
#define SQL_TRIM_END_           2
#define SQL_TRIM_NONE_          3

/* format string for dates coming from SQL gateways */
#define SQL_DATETIME_FORMAT_ "yyyy-mm-dd HH:mi:ss"
/* same format, expressed in Oracle. */
#define SQL_ORACLE_DATETIME_FORMAT_ "YYYY-MM-DD HH24:MI:SS"
```

DATA_APPLICATION_DLG@

Invokes Applixware Data

Format DATA_APPLICATION_DLG@([filename[, menubarID[, windowlessFlag[, title]]]])

Arguments

filename	The name of the file that will be loaded when the window is opened. (This argument is optional.)
menubarID	The number of a menu bar to be associated with this window. (This argument is optional.) This number should be between 200 and 299.
windowlessFlag	A Boolean value where TRUE indicates that no window will be displayed. FALSE is the default.
title	An optional title for the window. This value will replace the file name that usually appears at the top of the window.

Description Invokes Applixware Data. Using this macro, you can optionally:

- Load a file into the Data window.
- Replace the default menu bar with one of your choosing.
- Run Data in the background.
- Give the Data window a new title.

DATA_TYPE@

Returns the data type of the passed element

Format dataType = DATA_TYPE@(datum)

Arguments datum The data type that is returned.

Description Returns the data type of the passed element. datum is one of the following:

0	Type not yet determined
1	number
2	text
5	built-in function
6	recordable C call
7	function key descriptor
8	array
9	variable data type
10	argument data type
11	global variable
12	extern variable
13	non-recorded function data type
14	recordable function
15	non-recordable top-level function
16	remove function data type
17	functions invokable by command key
18	label data type
19	integer data type
21	object
22	binary data
23	write method
24	read method
25	object local data variable

DATE@

Converts a complete year to a serial number

Format DATE@(year, month, day)

Arguments

year	A whole number between 0 and 99.
month	A whole number between 1 and 12.
day	A whole number between 1 and 31.

Description The DATE@ function converts a complete date (year, month, day) into a serial number. For example:

DATE@(84,8,6) returns 30900
DATE@(83,7,9) returns 30506
DATE@(85,3,9) returns 31115

Applixware checks the number you enter to make sure it is a valid entry. For example, if you enter DATE@(84,22,3), your cell displays ERROR because there is no 22nd month.

The DATE@ function is most commonly used to perform arithmetic operations involving date values. The formula DATE@(84,8,6)-DATE@(83,7,9) calculates the number of elapsed days between July 9, 1983 and August 6, 1984. It returns the value 394.

DATEVALUE@

Returns the number of days since 12/31/1899

Format DATEVALUE@(dateString)

Arguments

dateString	A string containing a formatted date.

Description DATEVALUE@ returns a number representing the dateString argument. The number returned is the total number of days since 12/31/1899. For example, DATEVALUE@("1/1/1900") is 1, DATEVALUE@("1/2/1900") is 2, and so on.

The dateString is a formatted date. For a list of valid date formats, choose Style Ý Numbers from Applixware Spreadsheets and use one of the formats displayed in the scrolling list when you click on Date.

If the date format is ambiguous, such as "10/07/95" which could mean either October 7 or July 10, the first date format in the Number Style list is used.

Time information in dateString is ignored.

DATESTR@

Returns a formatted date string

Format DATESTR@(dateNumber, format, fullYearFlag)

Arguments

dateNumber — A date value as returned by the macro DATEVALUE@. This is the number of days after 12/31/1899 that the given date falls.

format — a number from 1 to 20 indicating the format of the returned string.

Format Value	Result
1	May 18, 1996
2	May 18, 1996
3	18 May 1996
4	05/18/1996
5	18.05.1996
6	1996-05-18
7	1996-05-18
8	1996 05 18
9	1996 05 18
10	19960518
11	19960518
12	18/05/1996
13	18.05.1996
14	May 18,1996
15	May 1996
16	May 1996
17	May 1996
18	05/18
19	1996 05
20	1996 05

fullYearFlag — if TRUE, display the year as a four digit year. For example, 05/18/87 is displayed as 05/18/1987 if the fullYearFlag is TRUE.

Description Returns a formatted date string. The format of the string is determined by the format variable. The year can be in either a two-digit (87) or four digit (1987) format.

DATETIME_CHANGE_FORMAT@

Changes a formatted time/date string to another format

Format DATETIME_CHANGE_FORMAT@(dtString, dtFormat)

Arguments

dtString	A formatted datetime string
dtFormat	A string or number indicating the new format for the datetime information.

Description Converts a formatted datetime string into a datetime string with a different format. The dtFormat argument can be either a string, or a number, as defined in the file datetime_.sp.

The example below, retrieves the current date and time, and displays the information in two different formats.

```
macro test
var x, y

x = datetime_now@()
info_message@("The date: " ++   DATETIME_TO_STRING@(x, 1))
' Prints 'The date: August 9, 1996'
y = DATETIME_CHANGE_FORMAT@(x, "Mm d, yyyy")
' Prints 'The date: Aug 9, 1996'
info_message@("The date: " ++ y)
endmacro
```

See also **Date Formats** lists the numbers and strings that are used in the dtFormat argument.

DATETIME_NOW@

Return the current date and time in an array

Format DATETIME_NOW@()

Description Returns an array containing a datetime_ format. This format is defined in the file **datetim_.am**. Seconds are rounded to 4 second intervals.

DATETIME_TO_STRING@

Converts a datetime_ array to a string

Format DATETIME_TO_STRING@(format datetime_ dtArray, dtFormat)

Arguments

dtArray — A datetim_ array, as defined in **datetim_.am**.

dtFormat — A string or number indicating the format for the datetime information.

Description Converts a datetim_ array into a string. The dtFormat argument can be either a string, or a number, as defined in the file datetime_.sp. Several examples follow:

```
DATETIME_TO_STRING@(array, 1)          ' August 9, 1996
DATETIME_TO_STRING@(array, "Mmmm d, yyyy")
                                       ' August 9, 1996

DATETIME_TO_STRING@(array, 2)          ' Aug 9, 1996
DATETIME_TO_STRING@(array, "Mm d, yyyy")
                                       ' Aug 9, 1996

DATETIME_TO_STRING@(array, 4)          ' 08/09/96
DATETIME_TO_STRING@(array, "mm/dd/yy")
                                       ' 08/09/96
```

See also **Date Formats** lists the numbers and strings that are used in the dtFormat argument.

DATESTR@

DATETIME_STRING_TO_DATETIME@

Converts a formatted
string to a datetime_ array

Format DATETIME_STRING_TO_DATETIME@(dtString)

Arguments

dtString — A string or number indicating the format for the datetime information.

Description Converts a formatted string into a datetim_ array. The datetim_ array is defined in the ELF include file datetime_.am.

See also Date Formats lists the possible formats for the dtString argument.

DATETIME_STRING_TO_TIME_VALUE@

Returns the UNIX time

Format DATETIME_STRING_TO_TIME_VALUE@(dtString)

Arguments

dtString	A formatted datetime string

Description Returns (18000 + the number of seconds since January 1, 1970). This macro is used to fill the EDAT, DDAT, RDAT fields in the OM_SEND_MESSAGE@ macro.

See also OM_SEND_MESSAGE@

DAYS360@

Returns the number of days between two dates

Format DAYS360@(start_date, end_date, method)

Arguments

start_date	Start date of the range of days to be measured.
end_date	End date of the range of days to be measured.
method	A boolean. If TRUE, use the US method to measure the number of days. If FALSE, use the European method.

Description DAYS360@ returns the number of days between two dates based on a 360-day year (twelve 30-day months). This artificial 360-day year is routinely used in the securities industry.

The arguments start_date and end_date can be either text strings that represent the month, day, and year, such as "10/24/95", or serial numbers representing the dates.

The method is a logical argument specifying whether to use the U.S. or European method to calculate the function.

Method	Definition
FALSE or omitted	US (NASD). If the starting date is the 31st of a month, it becomes equal to the 30th of the same month. If the ending date is the 31st of a month and the starting date is less than the 30th of a month, the ending date becomes equal to the 1st of the next month; otherwise the ending date becomes equal to the 30th of the same month.
TRUE	European method. Starting dates or ending dates that occur on the 31st of a month become equal to the 30th of the same month.

For example, DAYS360@("1/30/95", "2/1/95") equals 1.

DATE_FORMAT@

Converts a UNIX time/date value to a time/date string

Format string = DATE_FORMAT@(date, format)

Arguments

date	The UNIX date/time value, indicating the number of seconds elapsed since January 1, 1970, GMT. Use CURRENT_TIME@ to get this value.
format	A number or combination of numbers indicating how the date/time string should be represented.

Description Returns a string version of the current date and/or time. Daylight savings time is considered. See also **CURRENT_TIME@**. The following is a list of the date/time formats. You can combine elements from the table together to form a unified date/day/time format. For example, a valid format might be the number 4202.

NOTE: The following list approximates what you may receive. The **datetime.sp** file contains a description of exactly what will be returned.

Date Formats

#	Format	Description
1.	Mmmm d, yyyy	December 2, 1992 or December 12, 1992
2.	Mm d, yyyy	Dec 2, 1992 or Dec 12, 1992
3.	d Mm yyyy	2 December 1992 or 12 December 1992
4.	mm/dd/yy	12/02/92
5.	dd.mm.yy	02.12.92

6.	yyyy-mm-dd	1992-12-02
7.	yy-mm-dd	92-12-02
8.	yyyy mm dd	1992 12 02
9.	yy mm dd	92 12 02
10.	yyyymmdd	19921202
11.	yymmdd	921202
12.	dd/mm/yy	02/12/92
13.	dd.mm.yyyy	02.12.92
14.	Mm dd, yyyy	Dec 02, 1992 (the Inbox date format)
15.	Mmmm yyyy	December 1992
16.	Mm yyyy	Dec 1992
17.	Mm yy	Dec 92
18.	mm/dd	12/02
19.	yy mm	92/12
20.	yyyy mm	1992/12

Time formats:

#	Format	Description		
100	hh:mi pm		09:34 pm	12-hour format
200	hh:mipm		09:34pm	12-hour format
300	HH:mi	21:34	24-hour format	
400	HH.mi	21.34	24-hour format	
500	HHmi	2134	24-hour format	
600	HH:mi:ss	21:34:05	24-hour format	

Day of the week formats:

0	No day of week
1000	Tue <date/time>
2000	<date/time> Tue
3000	Tuesday <date/time>
4000	<date/time> Tuesday

Example

DATE_LAST_MODIFIED@

Returns the time a file was last modified

Format time = DATE_LAST_MODIFIED@(name)

Arguments name The file's relative or absolute path name.

Description Returns a value representing the time a file was last modified. This time is based on the number of seconds elapsed since January 1, 1970 GMT. Returns the value 0 if the specified file does not exist.

DAY@

Extracts the day of the month from a serial date number

Format DAY@(dateNumber)

Arguments dateNumber A serial date number.

Description Extracts the day of the month (1-31) from a serial date number. You can enter a serial date number as an argument for the DAY@ function. A formula that contains the serial number 30899 (August 6, 1984) returns 6.

You can also use the **DATE@** or **TODAY@** function as an argument in the DAY@ function. For example, the formula DAY@(DATE@(84,1,3)) returns 3.

DB_ACCEPT_POKES@

Indicates which messages a dialog box macro will accept

Format DB_ACCEPT_POKES@(dbox, codeArray)

Arguments

dbox	The name of the dialog box variable.
codeArray	An array of poke message codes that will be accepted by the dialog box macro.

Description Used with DB_SEND_POKE@ and DB_GET_POKE@ to send messages between a task and a dialog box task. A dialog box macro will only respond to messages having poke codes assigned to it using DB_ACCEPT_POKES@.

See also **DB_GET_POKE@**
DB_SEND_POKE@

DB_CANCELLED@

Indicates whether a dialog box was cancelled

Format flag = DB_CANCELLED@(dbox)

Arguments dbox The name of the dialog box variable.

Description Returns TRUE if a user presses CANCEL or hits ESCAPE in a dialog box; otherwise it returns FALSE. Push buttons of type Dismiss cancel a dialog box when pressed.

DB_CLOSE@

Closes a dialog box

Format DB_CLOSE@()

Description Closes (removes) a dialog box. Execute & Dismiss and Dismiss push buttons automatically call DB_CLOSE@. Using DB_CLOSE@ to close a dialog box is only necessary when you want a dialog box to close without the use of Execute & Dismiss or Dismiss buttons.

See also DB_CANCELLED@

DB_CREATE_CTRL@

Creates a dialog box control or label

Format DB_CREATE_CTRL@(dbox, type, name, title, xpos, ypos, default)

Arguments dbox The name of the dialog box variable.

type One of the following control types:

0 radio button group
1 toggle button
2 option button
3 push button
4 entry box
5 label
6 bitmap (decoration)
7 list box
8 panel/line (decoration)
9 edit box (scrollable multi-line, word-wrapping, text-editing, control).
11 scale
12 table
13 row/col
14 canvas

name The string name of the control.

title — A string indicating the control title displayed with the control for push buttons, entry boxes, toggle buttons, radio groups, option buttons, and labels.

For bitmaps, title should be the bitmap file name, without the .im extension. The bitmap file must be in your macros directory.

xpos — A number indicating the x-axis position, in pixels, for the top left corner of the control. This position is relative to the top left corner of the dialog box.

ypos — A number indicating the y-axis position, in pixels, for the top left corner of the control. This position is relative to the top left corner of the dialog box.

default — The default selection or text, which depends on the control type you specify:

Entry boxes — A string indicating the text in the entry box. An empty string indicates an empty entry box.

Radio button groups and Option Buttons — A number indicating the radio button, option, or list box item that is selected. Items are numbered from 0. A value of -1 indicates that no item is selected.

List boxes — A string indicating the item that is selected in a list box. If the list box allows multiple selections, you can specify a string array to indicate the items that are selected.

Edit boxes — An array of strings representing the contents. If the string is only one line, it must be an array. That is, it assigns the string to array [0].

Push buttons controls, bitmaps, and panel/line decorations do not have values.

Description Dialog box controls are normally created using the Dialog Box Editor rather than using DB_CREATE_CTRL@.

If you are creating a radio button group, an option button, or a list box, define the items to be used with these controls using DB_CTRL_STRINGS@.

If you are creating an entry box, use DB_CTRL_WIDTH@ to specify the width of the entry box and DB_CTRL_LENGTH@ to indicate how many characters can be typed in the entry box.

If you are creating a list box or an edit box, use DB_CTRL_HEIGHT@ to specify the height, in number of lines, of the list box and DB_CTRL_WIDTH@ to indicate how many characters wide to make the list

If you are creating a panel/line decoration use the following:

DB_CTRL_HEIGHT@
Indicates the number of pixels high to make the panel/line.

DB_CTRL_WIDTH@
Indicates the number of pixels wide to make the panel/line.

DB_CTRL_LINE_THICKNESS@
Indicate the thickness of the lines in the panel/line decoration.

If you are creating a push button, you will need to use DB_CTRL_BUTTON_TYPE@ to specify the type of push button. For example, you need to indicate if the push button is Execute & Dismiss, Dismiss, Execute, Help, or Bitmap. For more information, see **DB_CTRL_BUTTON_TYPE@**.

Example

See also **DB_CTRL_STRINGS@**
DB_CTRL_WIDTH@
DB_CTRL_LENGTH@
DB_CTRL_HEIGHT@
DB_CTRL_LINE_THICKNESS@
DB_DESTROY_CTRL@

DIALOG_WINDOW_ID@

Returns the dialog handle of a window

Format DIALOG_WINDOW_ID@(name)

Arguments name The title of the dialog for which you want to find the id number.

Description Returns the dialog handle of the window whose name is name.

ODBC Interface

ELF includes a complete ODBC interface for database connectivity. These macros begin with the prefix ODBC and are derived from the Visigenic ODBC for UNIX Software Development Kit.

There are over 60 macros in the ELF ODBC family. Each of these macros calls an ODBC C-level function within Applixware. The C API is documented in the book *ODBC for UNIX SDK*, which is available from Visigenic. It is highly recommended that you purchase a copy of this book before you attempt to write ODBC applications with ELF.

ELF programmers should be aware of the following differences between the ELF ODBC interface and the VISIGENIC C-level interface:

- There is a direct, one-to-one correspondence between the ELF ODBC macros and the Visigenic C functions. For example, the macro ODBC_FETCH@ maps to the function **SQLFetch**.
- The arguments for the ELF ODBC macros correspond to the input parameters for the Visigenic functions.
- All ELF ODBC macros return an array containing three elements:

1. A return code
2. An array of results. The results returned correspond to the output parameters of the function.
3. Error information

- For ODBC_EXTENDED_FETCH@, or any other macro that supports setting the bind type, the ELF ODBC interface supports only column-wise binding. Row-wise binding is not supported.
- SQL Bookmarks are not supported.
- The following ODBC functions do not have an ELF equivalent:

ODBC_BROWSE_CONNECT@ ODBC_DRIVER_CONNECT@
ODBC_PARAM_OPTIONS@

ODBC_ALLOC_CONNECT@

Returns a connection handle

Format ODBC_ALLOC_CONNECT@(format sql_channel@ channel)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

Description Returns an array containing an ODBC connection handle and a return code. A *connection handle* references information such as the valid statement handles on a connection, and whether a transaction is currently open.

To extract the connection handle, which is used to call other ODBC macros such as ODBC_TRANSACT@ and ODBC_CONNECT@, use the following code:

retval = ODBC_ALLOC_CONNECT@(channel)

connectHandle = retval[1, 0]

During an ODBC session, you can have multiple connections open at the same time. For example, if you want to execute transactions with two different databases in the same session, you can call ODBC_ALLOC_CONNECT@ twice to receive two different connection handles.

This macro must be preceded by the macros ODBC_START_SERVER@ and ODBC_ALLOC_ENV@.

See also ODBC_ALLOC_ENV@, ODBC_START_SERVER@

ODBC_ALLOC_ENV@

Initializes the ODBC call Interface for an application

Format ODBC_ALLOC_ENV@(format sql_channel@ channel)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

Description If successful, this macro returns either SQL_SUCCESS or SQL_SUCCESS_WITH_INFO. These constants are defined in dbase_.am.

ODBC_ALLOC_ENV@ initializes the ODBC call interface for use by an application. An application must call ODBC_START_SERVER@ and ODBC_ALLOC_ENV@ before calling an other ODBC macros.

See also ODBC_START_SERVER@, ODBC_ALLOC_CONNECT@

ODBC_ALLOC_STMT@

Established an SQL statement handle

Format ODBC_ALLOC_STMT@(format sql_channel@ channel, dbHandle)

Arguments channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

dbHandle — A connection handle as returned by ODBC_ALLOC_CONNECT@

Description If successful, this macro returns array with two elements: a return code and a SQL statement handle. The return code is one of the following:

SQL_SUCCESS	SQL_INVALID_HANDLE
SQL_SUCCESS_WITH_INFO	SQL_ERROR

This statement handle is used to submit SQL statements for processing by the server. For example, you pass the SQL statement handle to the ODBC_TABLES@ macro to get a list of tables stored in a data source.

To use ODBC_ALLOC_STMT@, you must start the server and establish a connection to the database. To do this, call the following macros, in the order shown:

4. ODBC_START_SERVER@
5. ODBC_ALLOC_CONNECT@
6. ODBC_SET_CONNECT_OPTION@
7. ODBC_CONNECT@

See also ODBC_START_SERVER@, ODBC_ALLOC_CONNECT@, ODBC_SET_CONNECT_OPTION@, ODBC_CONNECT@

ODBC_BATCH@

Runs sets of ODBC macros

Format ODBC_BATCH@(format sql_channel@ channel, macroArray)

Arguments channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

macroArray — An array containing the macros that you want to run as a group.

Description ODBC_BATCH@ runs a set of macros, and returns an array containing the results of all of the macros. Each element of the macroArray argument contains the names of the macros that you want to run, and the arguments of that macro, in the following format:

macroname arg1, arg2, ... arg*n*

Execution of the array of macros is aborted when a macro throws an error.

ODBC_BIND_COL@

Specifies the columns in a result set to be bound and will be returned by ODBC_FETCH@

Format ODBC_BIND_COL@(format sql_channel@ channel, stmtHandle, icol, datatype, maxlength)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@

icol — column number of result data, ordered from left to right starting at 1. A column of number 0 is used to retrieve the bookmark for the row. Bookmarks are not supported by ODBC 1.0 drivers or by ODBC_FETCH@.

datatype — The data type of the result data

maxlength — the maximum possible length of the column data

Description Specifies the columns in a result set to be returned with ODBC_FETCH@, and binds those columns to storage space on the server so that ODBC_FETCH@ can access them.

The datatype argument is one of the following constants:

DataType	Description	Length
EO_BINARY	Binary data	2^{32} bytes
EO_INT	Signed long integer	4 bytes
EO_DATETIME	Character data holding a datetime string	30 bytes
EO_STRING	Character data	Defined length of the column

EO_BOOL	Contains TRUE or FALSE	1 byte
EO_FLOAT	Signed number value with precision 15.	8 bytes

Maxlength specifies the maximum possible length of the result data. For numberic values, the lengths are shown in the table. For string data, such as EO_STRING, the maximum length of data is 255.

ODBC_BIND_COL@ returns an array of 3 elements, as follows:

- return[0] is the return code
- return[1] is an array of results
- return[2] is error information

See also ODBC_FETCH@

ODBC_BIND_PARAMETER@

Bind a parameter marker
in an SQL statement

Format ODBC_BIND_PARAMETER@(format sql_channel@ channel, stmtHandle, iPar, ParamType, Ctype, SqlType, Precision, Scale, Valuemax)

Arguments

channel	an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information: host — A string containing the name of the ODBC gateway id — An Applixware task ID as returned by ODBC_START_SERVER@
stmtHandle	An SQL statement handle as returned by ODBC_ALLOC_STMT@.
iPar	Parameter number, ordered sequentially, left-to-right, starting at 1.
fParamType	The type of the parameter: either SQL_PARAM_INPUT, SQL_PARAM_OUPUT, or SQL_PARAM_INPUT_OUPUT.
fCtype	The C data type of the parameter.
fSqltype	The SQL data type of the parameter.
cBcoldef	The precision of the column.
iBscale	The scale of the column or expression of the corresponding parameter marker.
cBvaluemax	Maximum length of the data.

Description ODBC_BIND_PARAMETER@ is used to bind markers in a statement handle to memory buffers on the gateway. There are two ways to do this in the ELF ODBC interface. ODBC_BIND_PARAMETER@ is used for small data types. ODBC_BIND_PARAMETER_DATA_AT_EXEC@ is used for large data types. (The definitions of "small" and "large" are left to the discretion of the programmer.)

For ODBC_BIND_PARAMETER@, the order of macro calls is as follows:

1. ODBC_PREPARE@ - Prepare a SQL statement with parameter markers. This returns a statement handle.
2. ODBC_BIND_PARAMETER@ - Allocate memory for data corresponding to the parameter markers in the prepared SQL statement.

   Loop
3. ODBC_BIND_VALUES@ - specify values to assign to the parameter markers. These values are written to the buffers allocated by the ODBC_BIND_PARAMETER@ statement.
4. ODBC_EXECUTE@ - execute the SQL statement with the parameter marker values assigned by ODBC_BIND_VALUES@.

   end Loop

fCtype is one of the following parameters:

DataType	Description	Maximum Length
EO_BINARY	Binary data	2^{32} bytes
EO_INT	Signed long integer	4 bytes
EO_DATETIME	Character data holding a datetime string	30 bytes
EO_STRING	Character data	Defined length of the column
EO_BOOL	Contains TRUE or FALSE	1 byte
EO_FLOAT	Signed number value with precision 15.	8 bytes

fSqltype is one of the following parameters:

SQL_BIGINT	SQL_DECIMAL
SQL_BINARY	SQL_DOUBLE
SQL_BIT	SQL_FLOAT
SQL_CHAR	SQL_INTEGER
SQL_DATE	SQL_LONGVARBINARY
SQL_LONGVARCHAR	SQL_TIMESTAMP
SQL_NUMERIC	SQL_TINYINT

SQL_REAL	SQL_VARBINARY
SQL_SMALLINT	SQL_VARCHAR
SQL_TIME	

Fsqltype could also be a driver-specific value.

The cBcoldef argument specifies the precision of the column or expression corresponding to the parameter marker, unless both of the following are TRUE:

- The fSqlType argument is SQLLONGVARBINARY or SQLLONGVARCHAR.
- The data for the parameter will be sent out with ODBC_PUT_DATA@.

cbValueMax depends on the type of data being added to the database. The discussion of fCtype has suggested maximum values for each data type.

See also ODBC_BIND_PARAMETER_DATA_AT_EXEC@

ODBC_BIND_PARAMETER_DATA_AT_EXEC@

Bind a parameter marker in an SQL statement

Format ODBC_BIND_PARAMETER_DATA_AT_EXEC@(format sql_channel@ channel, stmtHandle, iPar, ParamType, Ctype, SqlType, Precision, Scale, Valuemax)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

iPar — Parameter number, ordered sequentially, left-to-right, starting at 1.

ParamType — The type of the parameter: either SQL_PARAM_INPUT, SQL_PARAM_OUPUT, or SQL_PARAM_INPUT_OUPUT.

Ctype — The C data type of the parameter.

Sqltype — The SQL data type of the parameter.

Precision — The precision of the column.

Scale — The scale of the column or expression of the corresponding parameter marker.

Valuemax Maximum length of the data.

Description ODBC_BIND_PARAMETER_DATA_AT_EXEC@ is used to allocate memory for data that is added to a table in a database. There are two ways to do this in the ELF ODBC interface. ODBC_BIND_PARAMETER@ is used for small data types.

ODBC_BIND_PARAMETER_DATA_AT_EXEC@ is used for large data types. (The definitions of "small" and "large" are left to the discretion of the programmer.)

For ODBC_BIND_PARAMETER_DATA_AT_EXEC@, the order of macro calls is as follows:

1. ODBC_PREPARE@ - Prepare a SQL statement with parameter markers. This returns a statement handle.
2. ODBC_BIND_PARAMETER_DATA_AT_EXEC@ tells the driver what type of data to expect from the impending ODBC_PUT_DATA@.
3. ODBC_EXECUTE@. This macro returns SQL_NEED_DATA.

Loop

4. Call ODBC_PARAM_DATA@ to retrieve the token for the parameter marker. If there are two parameter markers in the SQL statement, the first parameter marker token is returned by the first ODBC_PARAM_DATA@ macro, the second is returned by the second ODBC_PARAM_DATA@ macro, and so on. When data has been supplied for all of the parameter markers in an SQL statement, ODBC_PARAM_DATA@ returns SQL_SUCCESS.
5. Call ODBC_PUT_DATA@ one or more times to supply the data for the parameter marker. When all the data for a parameter marker has been sent, call ODBC_PARAM_DATA@ a second time to retrieve the next parameter marker token.

end Loop

Ctype is one of the following parameters:

DataType	Description	Maximum Length
EO_BINARY	Binary data	2^{32} bytes
EO_INT	Signed long integer	4 bytes
EO_DATETIME	Character data holding a datetime string	30 bytes
EO_STRING	Character data	Defined length of the column
EO_BOOL	Contains TRUE or FALSE	1 byte
EO_FLOAT	Signed number value with precision 15.	8 bytes

Sqltype is one of the following parameters:

SQL_BIGINT SQL_DECIMAL

SQL_BINARY	SQL_DOUBLE
SQL_BIT	SQL_FLOAT
SQL_CHAR	SQL_INTEGER
SQL_DATE	SQL_LONGVARBINARY
SQL_LONGVARCHAR	SQL_TIMESTAMP
SQL_NUMERIC	SQL_TINYINT
SQL_REAL	SQL_VARBINARY
SQL_SMALLINT	SQL_VARCHAR
SQL_TIME	

Sqltype could also be a driver-specific value.

The Precision argument specifies the precision of the column or expression corresponding to the parameter marker, unless both of the following are TRUE:

- The fSqlType argument is SQLLONGVARBINARY or SQLLONGVARCHAR.
- The data for the parameter will be sent out with ODBC_PUT_DATA@.

ValueMax depends on the type of data being added to the database. The discussion of Ctype has suggested maximum values for each data type.

See also ODBC_BIND_PARAMETER@

ODBC_BIND_VALUES@

Assigns values to parameter markers

Format ODBC_BIND_VALUES@(format sql_channel@ channel, stmtHandle, parameter, value)

Arguments

channel	an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information: host — A string containing the name of the ODBC gateway id — An Applixware task ID as returned by ODBC_START_SERVER@
stmtHandle	An SQL statement handle as returned by ODBC_ALLOC_STMT@.
parameter	Parameter number, ordered sequentially left-to-right, starting at 1.
value	Value assigned to the parameter

Description ODBC_BIND_VALUES@ assigns a single value to a parameter marker in a prepared SQL statement.

This macro allows one parameter to be allocated at a time. If the SQL statement has more than one parameter marker, you must call ODBC_BIND_VALUES@ in a loop to assign a value to each parameter marker in the statement.

More information on the proper use of ODBC_BIND_VALUES@ can be found in the **ODBC_BIND_PARAMETER@** help topic.

See also **ODBC_PREPARE@**, **ODBC_EXECUTE@**

ODBC_CANCEL@

Cancels processing on a statement handle

Format ODBC_CANCEL@(format sql_channel@ channel, stmtHandle)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

Description ODBC_CANCEL@ can cancel requests on a stmtHandle that need data. After ODBC_EXECUTE@ or ODBC_EXECUTE_DIRECT@ returns SQL_NEED_DATA and before data has been sent for all data-at-execution parameters, an application can call ODBC_CANCEL@ to cancel the statement execution. After the statement has been cancelled, the application can call ODBC_EXECUTE@ or ODBC_EXECUTE_DIRECT@ again. For more information, see ODBC_BIND_PARAMETER@.

After ODBC_SET_POS@ returns SQL_NEED_DATA and before data has been sent for all data-at-execution parameters, an application can call ODBC_CANCEL@ to cancel the operation. After the statement has been cancelled, the application can call ODBC_SET_POS@ again. Cancelling does not affect the cursor state or the current cursor position. For more information, see ODBC_SET_POS@.

See also **ODBC_BIND_PARAMETER@**, **ODBC_EXECUTE@**, **ODBC_SET_POS@**

ODBC_COL_ATTRIBUTES@

Returns information about a column in a result set

Format ODBC_COL_ATTRIBUTES@((format sql_channel@ channel, stmtHandle, icol, infoType)

Arguments

channel	an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:	
	host	A string containing the name of the ODBC gateway
	id	An Applixware task ID as returned by ODBC_START_SERVER@
stmtHandle	An SQL statement handle as returned by ODBC_ALLOC_STMT@.	
icol	column number of result data, ordered from left to right starting at 1. A column of number 0 is used to retrieve the bookmark for the row. Bookmarks are not supported by ODBC 1.0 drivers or by ODBC_FETCH@.	
infoType	The type of information that you want returned by the macro	

Description Returns an array of three elements, as follows:

- return[0] is the return code
- return[1] is an array of results. The information returned depends on the requested infoType.
- return[2] is error information

The infoType argument can be any one of the following constants:

SQL_COLUMN_AUTO_INCREMENT

SQL_COLUMN_CASE_SENSITIVE

SQL_COLUMN_COUNT

SQL_COLUMN_DISPLAY_SIZE

SQL_COLUMN_LABEL

SQL_COLUMN_LENGTH

SQL_COLUMN_NAME

SQL_COLUMN_NULLABLE

SQL_COLUMN_OWNER_NAME

SQL_COLUMN_QUALIFIER_NAME

SQL_COLUMN_SCALE
SQL_COLUMN_SEARCHABLE
SQL_COLUMN_TABLE_NAME
SQL_COLUMN_TYPE
SQL_COLUMN_TYPE_NAME
SQL_COLUMN_UNSIGNED
SQL_COLUMN_UPDATEABLE

ODBC_COLUMN_PRIVILEGES@

Returns a list of columns and associated privileges for the specified table

Format ODBC_COLUMN_PRIVILEGES@(format sql_channel@ channel, stmtHandle, tableQualifier, tableOwner, tableName, colName)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

tableQualifier — Qualifier name. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, and empty empty string ("") denotes those tables that do not have qualifiers.

tableOwner — Owner name. If a driver supports owners for some tables but not for others, such as when a driver retrieves data from different DBMSs, and empty empty string ("") denotes those tables that do not have owners.

tableName — Table name.

ColName — Search string pattern for column names

Description ODBC_COLUMN_PRIVILEGES@ returns a standard result set, ordered by TABLE_QUALIFIER, TABLE_OWNER, TABLE_NAME, COLUMN_NAME, and privilege. The columns in the result set are as follows:

TABLE_QUALIFIER is the Table qualifier identifier. This is NULL if not applicable to the data source. If a driver supports qualifiers for some tables and not for others, such as

when the driver retrieves data from different DBMSs, it returns an empty string "" for those tables that do not have qualifiers.

TABLE_OWNER is the Table owner identifier. This is NULL if not applicable to the data source. If a driver supports qualifiers for some tables and not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have owners.

TABLE_NAME is the table identifier.

COLUMN_NAME is the column identifier.

GRANTOR is the identifier of the user who granted the privilege. This is NULL if not applicable to the data source.

GRANTEE is the identifier of the user to whom the privilege was granted.

PRIVILEGE identifies the column privilege. This may be one of the following values:

- SELECT means the grantee is permitted to retrieve data for the column
- INSERT means the grantee is permitted to provide data for the column in new rows that are inserted into the associated table.
- UPDATE means the grantee is permitted to update data in the column.
- REFERENCE means the grantee is permitted to refer to the column within a constraint (for example, a unique, referential, or table check constraint).

IS_GRANTABLE indicates whether the grantee is permitted to grant the privileges to other users; "YES", "NO", or NULL if unknown or not applicable to the data source.

To retrieve information in a result set, use ODBC_BIND_COL@ and ODBC_FETCH@.

See also ODBC_COLUMNS@

ODBC_COLUMNS@

Lists columns in a table

Format ODBC_COLUMNS@(format sql_channel@ channel, stmtHandle, tableQualifier, tableOwner, tableName, columnType)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

tableQualifier Qualifier name. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, and empty empty string ("") denotes those tables that do not have qualifiers.

tableOwner Table Owner. If a driver supports owners for some tables but not for others, such as when a driver retrieves data from different DBMSs, and empty empty string ("") denotes those tables that do not have owners.

tableName string search pattern for table names.

columnName String search pattern for column names.

Description This macros is typically used before statement execution to retrieve information for a table or tables from the data source's catalog.

ODBC_COLUMNS@ returns the results as a standard result set, ordered by TABLE_QUALIFIER, TABLE_OWNER, and TABLE_NAME. The following lists the columns in the result set:

- **TABLE_QUALIFIER** is a table qualifier identifier. This is NULL if not applicable to the data source. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty empty string ("") denotes those tables that do not have qualifiers.
- **TABLE_OWNER** is a table owner identifier. This is NULL if not applicable to the data source. If a driver supports owners for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty empty string ("") denotes those tables that do not have owners.
- **TABLE_NAME** is a table identifier.
- **COLUMN_NAME** is a column identifier.
- **DATA_TYPE** is a SQL data type. This can be an ODBC SQL data type or a driver-specific SQL data type.
- **TYPE_NAME** is a data-source dependent data type name; for example, "CHAR", "VARCHAR", "MONEY", "LONG VARBINARY", or "CHAR () FOR BIT DATA".
- **PRECISION** is the precision of a column on the data source.
- **LENGTH** is the length, in bytes, of data transferred on an ODBC_FETCH@ or ODBC_GET_DATA@ operation if SQL_C_DEFAULT is specified. For numeric data, this size may be different from the size of the data stored on the data source. This value is the same as the **PRECISION** column for character or binary data.
- **SCALE** is the scale of the column on the data source. NULL is returned for data types where scale is not applicable.
- **RADIX** is for numeric data types. This field is either "10" or "2". **10** indicates that the values in PRECISION and SCALE give the number of decimal digits allowed for the column. For example, a

DECIMAL(12,5) column would return a **RADIX** of 10, a **PRECISION** of 12, and a **SCALE** of 5; A **FLOAT** column could return a **RADIX** of 10, a **PRECISION** of 15, and a **SCALE** of NULL.

2 indicates that the values in **PRECISION** and **SCALE** give the number of bits allowed in the column. For example, a **FLOAT** column could return a **RADIX** of 2, a **PRECISION** of 52, and a **SCALE** of NULL.

NULL is returned for all data types where radix is not applicable.

- **NULLABLE** is one of the following values:

  SQL_NO_NULLS means the column does not accept NULL values.

  SQL_NULLABLE meams the column accepts NULL values.

  SQL_NULLABLE_UNKNOWN meams the column accepts NULL values.

- **REMARKS** contains a text description of the column.

See also ODBC_BIND_COL@, ODBC_COLUMN_PRIVILEGES@

ODBC_CONNECT@

Establishes a connection to a data source

Format ODBC_CONNECT@(format sql_channel@ channel, db_handle, source, uid, passwd)

Arguments channel an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

host A string containing the name of the ODBC gateway

id An Applixware task ID as returned by ODBC_START_SERVER@

db_handle A connection handle as returned by ODBC_ALLOC_CONNECT@. To extract the connection handle, use the following code:

```
retval = ODBC_ALLOC_CONNECT@(channel)
dbHandle = retval[1, 0]
```

source One of the data sources specified in the ODBC Data Sources section of the odbc.ini file.

uid a valid user ID for the data source

password A valid password for the user ID

Description Loads a driver and established a connection to the data source. The connection handle references strage of all information about the connection, including status, transaction state, and error information.

See also ODBC_ALLOC_CONNECT@

ODBC_DATA_SOURCES@

Lists data source names

Format ODBC_DATA_SOURCES@(format sql_channel@ channel, direction)

Arguments channel an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

host A string containing the name of the ODBC gateway

id An Applixware task ID as returned by ODBC_START_SERVER@

Direction Determines whether the Driver Manager fetches the next driver description in the list (SQL_FETCH_NEXT) or whether the search starts from the beginning of the list (SQL_FETCH_FIRST).

Description ODBC_DATA_SOURCES@ is implemented by the Data Manager. This macro returns an entry from the **DRIVERS** section of the odbc.ini file in your HOME@ directory.

Because it is implemented in the Driver Manager, ODBC_DATA_SOURCES@ is supported for all drivers, regardless of a particular driver's conformance level.

An application can call ODBC_DATA_SOURCES@ multiple times to retrieve all data source names. The Driver Manager retrieves this information from the **odbc.ini** file. When there are no more data sources names, the Driver Manager returns SQL_NO_DATA_FOUND. If ODBC_DATA_SOURCES@ is called with SQL_FETCH_NEXT immediately after it returns SQL_NO_DATA_FOUND, it will return the first data source name.

If SQL_FETCH_NEXT is passed to ODBC_DATA_SOURCES@ the very first time it is called, it returns the first data source name.

The driver determines how data source names are mapped to actual data sources.

See also ODBC_CONNECT@, ODBC_DRIVERS@

ODBC_DESCRIBE_COL@

Returns information on a column in a result set

Format ODBC_DESCRIBE_COL@(format sql_channel@ channel, stmtHandle, iCol)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

iCol — A column number in the result set.

Description An application typically calls ODBC_DESCRIBE_COL@ after a call to ODBC_PREPARE@ and before or after the associated call to ODBC_EXECUTE@. An application can also call ODBC_DESCRIBE_COL@ after a call to ODBC_EXEC_DIRECT@.

ODBC_DESCRIBE_COL@ retrieves the column name, type, and length generated by a **SELECT** statement. If the column is an expression, the returned column name is either an empty string or a driver-defined name.

ODBC_DESCRIBE_PARAM@

Returns information about a parameter marker in a SQL statement

Format ODBC_DESCRIBE_PARAM@(format sql_channel@ channel, stmtHandle, iPar)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

iPar — Parameter marker number ordered sequentially from left to right in the prepared SQL statement.

Description Returns a description of a parameter marker in a prepared SQL statement. Parameter markers are numbered from left to right in the order they appear in the SQL statement.

ODBC_DESCRIBE_PARAM@ does not return the type (input, input/output, or output) of a parameter in an SQL statement. Except in calls to procedures, all parameters in SQL statements are input parameters. To determine the type of each parameter in a call to a procedure, an application calls ODBC_PROCEDURE_COLUMNS@.

See also ODBC_PROCEDURE_COLUMNS@, ODBC_PREPARE@, ODBC_BIND_PARAMETER@

ODBC_DISCONNECT@

Terminates a connection to a data source

Format ODBC_DISCONNECT@(format sql_channel@ channel, db_handle)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

db_handle A connection handle as returned by ODBC_ALLOC_CONNECT@

Description If an application calls ODBC_DISCONNECT@ while there is an incomplete transaction associated with the connection handle, the driver returns SQLSTATE 25000 (Invalid transaction state), indicating that the transaction is unchanged and the connection is open. An incomplete transaction is one that has not been committed or rolled back with ODBC_TRANSACT@.

If an application calls ODBC_DISCONNECT@ before is has freed all statement handles associated with the connection, the driver frees those statement handles after it successfully disconnects from the data source. However, if one of the statement handles associated with the connection are still executing asynchronously, ODBC_DISCONNECT@ returns SQL_ERROR with SQLSTATE value of S1010 (Function sequence error).

See also ODBC_ALLOC_CONNECT@, ODBC_CONNECT@

ODBC_DRIVERS@

Lists driver descriptions and driver attribute keywords

Format ODBC_DRIVERS@(format sql_channel@ channel, Direction)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

Direction — Determines whether the Driver Manager fetches the next driver description in the list (SQL_FETCH_NEXT) or whether the search starts from the beginning of the list (SQL_FETCH_FIRST).

Description **ODBC_DRIVERS@** returns an array of three elements as follows:

- return[0] is the return code.
- return[1] is one line of the driver descriptions from the odbcinst.ini file on your local machine. If Direction = SQL_FETCH_FIRST, the first line of te driver descriptions is listed in return[1]. If Direction = SQL_FETCH_NEXT, the next line of the driver descriptions is listed. The following is an example of what the driver descriptions might look like:

```
[ODBC Drivers]
Visigenic MS SQL Server=Installed
Visigenic Sybase DBLib=Installed
```

Visigenic Sybase SQL Server 10=Installed

- return[2] is error information

See also ODBC_DATA_SOURCES@

ODBC_ERROR@

Returns error or status information

Format ODBC_ERROR@(format sql_channel@ channel, stmtHandle, dbHandle)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

dbHandle — A connection handle as returned by ODBC_ALLOC_CONNECT@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

Description An application typically calls ODBC_ERROR@ when a previous call to an ODBC macro returns SQL_ERROR or SQL_SUCCESS_WITH_INFO. However, any ODBC function can post zero or more errors each time it is called, so an application can call ODBC_ERROR@ after each ODBC macro call.

ODBC_ERROR@ returns an array of three elements:

- return[0] is the return code
- return[1] is an array of results. The information returned depends on the arguments supplied, as described below. Typically, and error code and an error string is present in this array. These are described in the help topic ODBC errors.
- return[2] is error information

ODBC_ERROR@ retrieves an error from the data structure associated with the right-most non-null handle argument. An application requests error information as follows:

- To retrieve errors associated with an environment, the application passes SQL_NULL_HDBC and SQL_NULL_HSTMT in dbHandle and stmtHandle respectively. The driver returns the error status of the ODBC macro called with the same connection handle.
- To retrieve errors associated with a connection, the application passes the corresponding connection handle (dbHandle) plus a statement handle equal to SQL_NULL_HSTMT. The driver returns the error status of the ODBC macro called with the same connection handle.

- To retrieve errors associated with a statement, an application passes the corresponding stmtHandle. If the call to ODBC_ERROR@ contains a valid statement handle, the driver ignores the database handle argument. The driver returns the error status of the ODBC macro most recently called with the same statement handle.
- To retrieve multiple errors for a function call, an application calls ODBC_ERROR@ multiple times. For each error the driver returns SQL_SUCCESS and removes that error from the list of available errors.

Where there is no additional information for the rightmost non-null handle, ODBC_ERROR@ returns SQL_NO_DATA_FOUND.

The Driver Manager stores error information in its in its environment, database, and statement handle structures. Similarly, the driver stores error information in its environment, database, and statement handle structures. When the application calls ODBC_ERROR, the driver manager checks if there are any errors in its structure for the specified handle. If there are errors for the specified handle, it returns the first error; if there are no errors, it calls ODBC_ERROR@ in the driver.

The Driver Manager can store up to 64 errors with an environment handle, and with its associated database and statement handles. When this limit is reached, the Driver Manager discards any subsequent errors posted on the Driver Manager's handles. The number of errors that a driver can store is driver-dependent.

An error ius removed from the structure associated with a handle when ODBC_ERROR@ is called for that handle and returns that error. All errors stored for a given handle are removed when that handle is used in a subsequent macro call. For example, errors on a statement handle that were returned by ODBC_EXEC_DIRECT@ are removed when ODBC_EXEC_DIRECT@ or ODBC_TABLES@ is called with that statement handle. The errors store on a handle are not removed as the result of a call to a macro using an associated handle of a different type. For example, errors on a database handle that were returned by ODBC_NATIVE_SQL@ are not removed when ODBC_ERROR@ or ODBC_EXEC_DIRECT@ are called on a statement handle associated with that database handle.

ODBC_EXECUTE@

Executes a prepared SQL statement

Format ODBC_EXECUTE@(format sql_channel@ channel, stmtHandle)

Arguments channel an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

host A string containing the name of the ODBC gateway

id An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle An SQL statement handle as returned by ODBC_ALLOC_STMT@.

Description Executes a prepared using the current values of the parameter marker variables, if any exist in the statement. This macro allows you to call the SQL statement multiple times with different parameter values. For fast, one-time execution of a SQL statement,

The statement executed must have been prepared with ODBC_PREPARE@. Once the application processes or discards the results from a call to ODBC_EXECUTE@, the application can call ODBC_EXECUTE@ again with new parameter values.

To execute a **SELECT** statement more than once, the application must call ODBC_FREE_STMT@ with the SQL_CLOSE parameter before re-issuing the **SELECT** statement.

If the data source is in manual-commit mode (requiring explicit transaction initiation), and a transaction has not already been initiated, the driver initiates a transaction before it sends a SQL statement.

If an application uses ODBC_PREPARE@ to prepare and ODBC_EXECUTE@ to submit a **COMMIT** or **ROLLBACK** statement, it will not be interoperable between DBMS products. To commit or rollback a transaction, call ODBC_TRANSACT@.

If ODBC_EXECUTE@ encounters a data-at-execution parameter, it returns SQL_NEED_DATA. The application sends the data using ODBC_PARAM_DATA@ and ODBC_PUT_DATA@.

See also **ODBC_PARAM_DATA@**

ODBC_PUT_DATA@

ODBC_EXEC_DIRECT@

Executes a SQL statement

Format ODBC_EXEC_DIRECT@(format sql_channel@ channel, stmtHandle, sqlString)

Arguments channel an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

host A string containing the name of the ODBC gateway

id An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle An SQL statement handle as returned by ODBC_ALLOC_STMT@.

sqlString SQL text string to be translated

Description The application calls ODBC_EXEC_DIRECT@ to send an SQL statement directly to the data source. The driver modifies the statement to use the form of SQL used by the data source, the submits it to the data source. In particular, the driver modifies the escape clauses used to define ODBC-specific SQL.

The application can include one or more parameter markers in the SQL statement. To include a parameter marker, the application embeds a question mark (?) into the SQL statementat the appropriate position.

If the SQL statement is a **SELECT** statement, and if the application called ODBC_SET_CURSOR_NAME@ to associate a cursor with a statement handle, then the driver uses the specified cursor. Otherwise, the driver generates a cursor name.

If the data source is in manual-commit mode (requiring explicit transaction initiation), and a transaction has not already been initiated, the driver initiates a transaction before it sends a SQL statement.

If an application uses ODBC_EXEC_DIRECT@ to submit a **COMMIT** or **ROLLBACK** statement, it will not be interoperable between DBMS products. To commit or rollback a transaction, call ODBC_TRANSACT@.

If ODBC_EXEC_DIRECT@ encounters a data-at-execution parameter, it returns SQL_NEED_DATA. The application sends the data using ODBC_PARAM_DATA@ and ODBC_PUT_DATA@.

See also **ODBC_PARAM_DATA@**,
ODBC_PUT_DATA@

ODBC_EXTENDED_FETCH@

Fetches multiple rows of data from a result set

Format ODBC_EXTENDED_FETCH@(format sql_channel@ channel, stmtHandle, fetchType, iRow)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

fetchType	Type of fetch.
iRow	A row number. The meaning of this argument varies with the fetchType. This argument is ignored if the fetchType is SQL_FETCH_NEXT, SQL_FETCH_PRIOR, SQL_FETCH_FIRST, or SQL_FETCH_LAST.

Description ODBC_EXTENDED_FETCH@ returns a rowset of data to the application. An application cannot mix calls to ODBC_EXTENDED_FETCH@ and ODBC_FETCH@ for the same cursor.

The difference between ODBC_EXTENDED_FETCH@ and ODBC_FETCH@ is that ODBC_FETCH@ returns only one row of data, while ODBC_EXTENDED_FETCH@ returns as many rows as you want. Also, with ODBC_EXTENDED_FETCH@, you can select rows before the cursor, or at the beginning or end of the result set.

The sequence of macros to call to return a set of rows is as follows:

1. Call ODBC_BIND_COL@ to specify the columns to return from the fetch.
2. Call ODBC_SET_STMT_OPTION@ with the SQL_ROWSET_SIZE option set to the number of rows you want returned.
3. Call ODBC_EXTENDED_FETCH@.

The fetchType argument must be set to one of the following constants:

SQL_FETCH_NEXT returns the next rowset. If the cursor is positioned before the start of the result set, this is equivalent to SQL_FETCH_FIRST.

SQL_FETCH_FIRST returns the first rowset in the result set.

SQL_FETCH_LAST returns the last rowset in the result set.

SQL_FETCH_PRIOR returns the prior rowset. If the cursor is positioned at the end of the result set, this is equivalent to SQL_FETCH_LAST.

SQL_FETCH_ABSOLUTE returns the rowset starting at row *irow*. If *irow* is equal to zero, the driver returns SQL_NO_DATA_FOUND and the cursor is positioned before the start of the result set.

If irow is less than 0, the driver returns the rowset starting at row n+*irow*+1, where n is the number of rows in the result set. For example, if irow is -1, the driver returns the rowset starting at the last row in the result set. If the result set size is 10 and irow is -10, the driver returns the rowset starting at the first row in the result set.

SQL_FETCH_RELATIVE returns the rowset *irow* rows from the start of the current rowset. If irow equals 0, the driver refreshes the current rowset.

SQL_FETCH_BOOKMARK is not currently supported by the ELF ODBC interface.

The ELF ODBC interface also does not support row-wise binding.

See also ODBC_FETCH@, ODBC_GET_STMT_OPTION@, ODBC_SET_STMT_OPTION@

ODBC_FETCH@

Fetches a row of data from a result set

Format ODBC_FETCH@(format sql_channel@ channel, stmtHandle)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

Description ODBC_FETCH@ positions the cursor on the next row of the result set. Before ODBC_FETCH@ is called for the first time, the cursor is positioned before the start of the result set. When the cursor is positioned on the last row of the result set, ODBC_FETCH@ returns SQL_NO_DATA_FOUND and the cursor is positioned after the end of the result set. An application cannot mix calls to ODBC_EXTENDED_FETCH@ and ODBC_FETCH@ for the same cursor.

If the application called ODBC_BIND_COL@ to bind columns, ODBC_FETCH@ stores data into the locations specified by the calls to ODBC_BIND_COL@. If the application does not call ODBC_BIND_COL@ to bind any columns, ODBC_FETCH@ returns no data, and moves the cursor to the next row in the result set.

An application can call ODBC_GET_DATA@ to retrieve data that is not bound to a storage location. ODBC_GET_DATA@ also allows you to retrieve data in increments. ODBC_FETCH@ only allows you to retrieve data from the start of the column.

For example, suppose you want to retrieve data from a 200-byte column. You can get all 200 bytes with ODBC_FETCH@, or you can get the first 100 bytes, then the second 100 bytes by calling ODBC_GET_DATA@ twice. ODBC_GET_DATA@ is commonly used to retrieve predictable amounts of data from a character column.

ODBC_FETCH@ accesses column data in left-to-right order, and is valid only after a call that returns a result set.

ODBC_FETCH@ returns an array of 3 elements, as follows:

- return[0] is the return code
- return[1] is an array of results. If the macro is successful, this array contains the requested row of data.
- return[2] is error information

See also ODBC_GET_DATA@

ODBC_FOREIGN_KEYS@

Returns a list of foreign keys

Format ODBC_FOREIGN_KEYS@(format sql_channel@ channel, stmtHandle, pTableQualifier, pTableOwner, pTableName, fTableQualifier, fTableOwner, fTableName)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

pTableQualifier — Primary key table qualifier. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty empty string ("") denotes those tables that do not have qualifiers.

pTableOwner — Primary key table owner. If a driver supports owners for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty empty string ("") denotes those tables that do not have owners.

pTableName — Primary key table name.

fTableQualifier — Foreign key table qualifier. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty empty string ("") denotes those tables that do not have qualifiers.

fTableOwner — Foreign key table owner. If a driver supports owners for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty empty string ("") denotes those tables that do not have owners.

fTableName — Foreign key table name.

Description ODBC_FOREIGN_KEYS@ can return:

- A list of foreign keys in the specified table (columns in the specified table that refer to primary keys in other tables.)

- A list of foreign keys in other tables that refer to the primary key in the specified table.

  A driver returns each list as a result set on the specified statement handle.

  If pTableName contains a table name, ODBC_FOREIGN_KEYS@ returns a result set containing the primary keys of the specified table and all the foreign keys that refer to it.

  If fTableName contains a table name, ODBC_FOREIGN_KEYS@ returns a result set containing all of the foreign keys in the specified table and the primary keys (in other tables) to which they refer.

  If both pTableName and fTableName contain table names, ODBC_FOREIGN_KEYS@ returns the foreign keys in the table specified in fTableName that refer to the primary key of the table specified in fTableName. This should be one key at most.

  ODBC_FOREIGN_KEYS@ returns a standard result set. If the foreign keys associated with a primary key are requested, the result set is ordered by FKTABLE_QUALIFIER, FKTABLE_OWNER, FKTABLE_NAME, and KEY_SEQ. If the primary keys associated with a foreign key are requested, the result set is ordered by PKTABLE_QUALIFIER, PKTABLE_OWNER, PKTABLE_NAME, and KEY_SEQ.

  The result set contains six columns, all of which are variable-length strings with a 128-byte maximum length unless otherwise noted. The columns are as follows:
- **PKTABLE_QUALIFIER** is a Primary key table qualifier identifier. This is NULL if not applicable to the data source. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty empty string ("") denotes those tables that do not have qualifiers.
- **PKTABLE_OWNER** is a Primary key table owner identifier. This is NULL if not applicable to the data source. If a driver supports owners for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty empty string ("") denotes those tables that do not have owners.
- **PKTABLE_NAME** is a Primary key table identifier.
- **PKCOLUMN_NAME** is a Primary key column identifier.
- **FKTABLE_QUALIFIER** is a foreign key table qualifier identifier. This is NULL if not applicable to the data source. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty empty string ("") denotes those tables that do not have qualifiers.
- **FKTABLE_OWNER** is a foreign key table owner identifier. This is NULL if not applicable to the data source. If a driver supports owners for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty empty string ("") denotes those tables that do not have owners.
- **FKTABLE_NAME** is a foreign key table identifier.
- **FKCOLUMN_NAME** is a Primary key column identifier.
- **KEY_SEQ** is the column sequence number in key, starting with 1. This data is an integer value.

- **UPDATE_RULE** is an action to be applied to the foreign key when the SQL operation is **UPDATE**:
  SQL_CASCADE SQL_RESTRICT SQL_SET_NULL
  This is NULL if not applicable to the data source.
- **DELETE_RULE** is an action to be applied to the foreign key when the SQL operation is **DELETE**:
  SQL_CASCADE SQL_RESTRICT SQL_SET_NULL
  This is NULL if not applicable to the data source.
- **FK_NAME** is a Foreign key identifier. This is NULL if not applicable to the data source.
- **PK_NAME** is a Primary key identifier. This is NULL if not applicable to the data source.

To retrieve information in a result set, use ODBC_BIND_COL@ and ODBC_FETCH@.

See also ODBC_FETCH@

ODBC_FREE_CONNECT@

Frees a connection handle

Format ODBC_FREE_CONNECT@(format sql_channel@ channel, dbHandle)

Arguments

channel	an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:	
	host	A string containing the name of the ODBC gateway
	id	An Applixware task ID as returned by ODBC_START_SERVER@
dbHandle	A connection handle as returned by ODBC_ALLOC_CONNECT@	

Description Frees the connection handle allocated by the macro ODBC_ALLOC_CONNECT@.

See also ODBC_ALLOC_CONNECT@

ODBC_FREE_ENV@

Frees the ODBC environment handle

Format ODBC_FREE_ENV@(format sql_channel@ channel)

Arguments

channel	an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

host A string containing the name of the ODBC gateway

id An Applixware task ID as returned by ODBC_START_SERVER@

Description If successful, this macro returns either SQL_SUCCESS or SQL_SUCCESS_WITH_INFO. These constants are defined in dbase_.am.

ODBC_FREE_ENV@ de-allocates the environment handle allocated by the ODBC_ALLOC_ENV@ macro.

See also ODBC_ALLOC_ENV@

ODBC_FREE_STMT@

Frees an SQL statement handle or cancels a request

Format ODBC_FREE_STMT@(format sql_channel@ channel, dbHandle, fOption)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@

foption — An outstanding request option. If this variable is set to SQL_DROP, the statement handle is de-allocated, and all requests using this statement handle are cancelled.

Description An application can call ODBC_FREE_STMT@ to terminate processing of a **SELECT** statement with or without de-allocating the statement handle.

Another way to cancel processing of a statement is to call ODBC_CANCEL@.

See also ODBC_ALLOC_STMT@, ODBC_CANCEL@

ODBC_GET_INFO@

Returns information about the ODBC driver and data source associated with a connection handle

Format ODBC_GET_INFO@(format sql_channel@ channel, dbHandle, infoType)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

dbHandle — A connection handle as returned by ODBC_ALLOC_CONNECT@

infoType — The type of information that you want returned by the macro

Description Returns many different kinds of information about the ODBC driver, or the data set associated with a connection handle.

The infoType argument is a constant defined in dbase_.am. Click one of the names below for more information:

SQL_ACCESSIBLE_PROCEDURES SQL_ACCESSIBLE_TABLES
SQL_ACTIVE_CONNECTIONS SQL_ACTIVE_STATEMENTS
SQL_ALTER_TABLE SQL_BOOKMARK_PERSISTANCE
SQL_COLUMN_ALIAS SQL_CONCAT_NULL_BEHAVIOR
SQL_CONVERT_* SQL_CONVERT_FUNCTIONS
SQL_CORRELATION_NAME SQL_CURSOR_COMMIT_BEHAVIOR
SQL_CURSOR_ROLLBACK_BEHAVIOR
SQL_DATA_SOURCE_NAME SQL_DATA_SOURCE_READ_ONLY
SQL_DATABASE_NAME SQL_DBMS_NAME
SQL_DBMS_VER SQL_DEFAULT_TXN_ISOLATION
SQL_DRIVER_HDBC SQL_DRIVER_HENV
SQL_DRIVER_HLIB SQL_DRIVER_HSTMT
SQL_DRIVER_NAME SQL_DRIVER_ODBC_VER
SQL_DRIVER_VER SQL_EXPRESSION_IN_ORDERBY
SQL_FETCH_DIRECTION SQL_FILE_USAGE
SQL_GETDATA_EXTENSIONS SQL_GROUP_BY
SQL_IDENTIFIER_CASE SQL_IDENTIFIER_QUOTE_CHAR
SQL_KEYWORDS SQL_LIKE_ESCAPE_CLAUSE
SQL_LOCK_TYPES SQL_MAX_BINARY_LITERAL_LEN
SQL_MAX_CHAR_LITERAL_LEN SQL_MAX_COLUMN_NAME_LEN

SQL_MAX_COLUMNS_IN_GROUP_BY
SQL_MAX_COLUMNS_IN_INDEX SQL_MAX_COLUMNS_IN_ORDER_BY
SQL_MAX_COLUMNS_IN_SELECT SQL_MAX_COLUMNS_IN_TABLE
SQL_MAX_CURSOR_NAME_LEN SQL_MAX_INDEX_SIZE
SQL_MAX_OWNER_NAME_LEN SQL_MAX_PROCEDURE_NAME_LEN
SQL_MAX_QUALIFIER_NAME_LEN SQL_MAX_ROW_SIZE
SQL_MAX_ROW_SIZE_INCLUDES_LONG
SQL_MAX_STATEMENT_LEN SQL_MAX_TABLE_NAME_LEN
SQL_MAX_TABLES_IN_SELECT
SQL_MULT_RESULT_SETS SQL_MULTIPLE_ACTIVE_TXN
SQL_NEED_LONG_DATA_LEN SQL_NON_NULLABLE_COLUMNS
SQL_NULL_COLLATION SQL_NUMERIC_FUNCTIONS
SQL_ODBC_API_CONFORMANCE SQL_ODBC_SAG_CLI_CONFORMANCE
SQL_ODBC_SQL_CONFORMANCE SQL_ODBC_SQL_OPT_IEF
SQL_ODBC_VER SQL_ORDER_BY_COLUMNS_IN_SELECT
SQL_OUTER_JOINS SQL_OWNER_TERM
SQL_OWNER_USAGE SQL_POS_OPERATIONS
SQL_POSITIONED_STATEMENTS SQL_PROCEDURE_TERM
SQL_PROCEDURES SQL_QUALIFIER_LOCATION
SQL_QUALIFIER_NAME_SEPARATOR
SQL_QUALIFIER_TERM SQL_QUALIFIER_USAGE
SQL_QUOTED_IDENTIFIER_CASE SQL_ROW_UPDATES
SQL_SCROLL_CONCURRENCY SQL_SCROLL_OPTIONS
SQL_SEARCH_PATTERN_ESCAPE SQL_SERVER_NAME
SQL_SPECIAL_CHARACTERS SQL_STATIC_SENSITIVITY
SQL_STRING_FUNCTIONS SQL_SUBQUERIES
SQL_SYSTEM_FUNCTIONS SQL_TABLE_TERM
SQL_TIMEDATE_ADD_INTERVALS SQL_TIMEDATE_DIFF_INTERVALS
SQL_TIMEDATE_FUNCTIONS SQL_TXN_CAPABLE
SQL_TXN_ISOLATION_OPTION SQL_UNION
SQL_USER_NAME

See also ODBC_ALLOC_CONNECT@, ODBC_START_SERVER@

ODBC_GET_CONNECT_OPTION@

Gets the parameters for an ODBC connection

Format ODBC_GET_CONNECT_OPTION@(format sql_channel@ channel, dbHandle, fOption)

Arguments channel an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

host A string containing the name of the ODBC gateway

id An Applixware task ID as returned by ODBC_START_SERVER@

dbHandle A connection handle as returned by ODBC_ALLOC_CONNECT@

fOption A constant value indicating the connection option to retrieve

Description If successful, this macro returns an array of two elements: the return code of the macro and the value of the requested connection parameter. These constants are defined in dbase_.am.

ODBC_SET_CONNECT_OPTION@ established parameters for a connection to a database.

The connection options that can be retrieved are as follows:

SQL_ACCESS_MODE	SQL_AUTOCOMMIT
SQL_LOGIN_TIMEOUT	SQL_OPT_TRACE
SQL_OPT_TRACEFILE	SQL_TXN_ISOLATION
SQL_CURRENT_QUALIFIER	SQL_ODBC_CURSORS
SQL_PACKET_SIZE	

These options are discussed in detail in the the discussion of the macro ODBC_SET_CONNECT_OPTION@.

See also ODBC_ALLOC_CONNECT@, ODBC_SET_CONNECT_OPTION@

ODBC_GET_CURSOR_NAME@

Returns the cursor name associated with a statement handle

Format ODBC_GET_CURSOR_NAME@(format sql_channel@ channel, stmtHandle)

Arguments channel an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

host A string containing the name of the ODBC gateway

id An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle An SQL statement handle as returned by ODBC_ALLOC_STMT@.

Description The only ODBC SQL statements that use a curesor name are positioned update and delete. For example:

UPDATE *table-name* ... **WHERE CURRENT OF** *cursor-name*

If the application does not call ODBC_SET_CURSOR_NAME@ to define a cursor name, on execution of a **SELECT** statement the driver generates a name that begins with the letters SQL_CUR and does not exceed 18 characters in length.

ODBC_GET_CURSOR_NAME@ returns the name of the cursor regardless of whether the name was created explicitly or implicitly.

A cursor name that is set explicitly or implicitly remain set until the associated statement handle is dropped, using ODBC_FREE_STMT@ with the SQL_DROP option.

See also ODBC_SET_CURSOR_NAME@

ODBC_GET_DATA@

Returns data for an unbound column

Format ODBC_GET_DATA@(format sql_channel@ channel, stmtHandle, icol, type, maxValue)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

iCol — Column number of result data, ordered sequentially from left to right, starting at 1. Retrieving bookmarks (column zero) is not supported.

type — The data type of the result data.

maxValue — Maximum length of the buffer allocated by the driver for the data type indicated by type.

Description Returns data for an unbound column in a table. If more than one call to ODBC_GET_DATA@ is required to retrieve data from a single column with a character, binary, or data-specific data type, the driver returns SQL_SUCCESS_WITH_INFO. A subsequent call to ODBC_ERROR@ returns SQLSTATE 01004 (Data truncated).

The application can then use the same column number to return subsequent parts of the data until ODBC_GET_DATA@ returns SQL_SUCCESS. This indicates that all data for the column has been retrieved.

ODBC_GET_DATA@ returns SQL_NO_DATA_FOUND when it is called for a column after all the data has been retrieved and before data has been retrieved for a subsequent column. The application can ignore excess data by proceeding to the next result column.

An application can use ODBC_GET_DATA@ to retrieve data from a column in parts only when retrieving character C data from a column with a character, binary, or data-specific data type or when retrieving binary C data from a column with a character, binary, or data-specific data type. If ODBC_GET_DATA@ is called more than one time in a row for a column under any other conditions, it returns SQL_NO_DATA_FOUND for all calls after the first.

For maximum interoperability, applications should call ODBC_GET_DATA@ only for unbound columns with numbers greater than the number of the last bound column. Within a single row of data, the column number in each successive call to ODBC_GET_DATA@ should be greater than or equal to the column number of the previous call. As extended functionality, drivers can return data through ODBC_GET_DATA@ from bound columns, from columns before the last bound column, or from columns in any order. To determine whether a driver supports these extensions, call ODBC_GET_INFO@ with the SQL_GETDATA_EXTENSIONS option.

Application that use ODBC_EXTENDED_FETCH@ should call ODBC_GET_DATA@ only when the rowset size is 1. As extended functionality, drivers can return data through ODBC_GET_DATA@ when the rowset size is greater than 1. The application calls ODBC_SET_POS@ to position the cursor on a row and calls ODBC_GET_DATA@ to retrieve data from an unbound column. To determine whether a driver supports these extensions, call ODBC_GET_INFO@ with the SQL_GETDATA_EXTENSIONS option.

See also ODBC_PUT_DATA@, ODBC_EXTENDED_FETCH@

ODBC_GET_FUNCTIONS@

Returns information about whether
a driver supports an ODBC function

Format ODBC_GET_FUNCTIONS@(format sql_channel@ channel, dbHandle, func)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

dbHandle	A connection handle as returned by ODBC_ALLOC_CONNECT@
func	A constant indicating the function to check

Description ODBC_GET_FUNCTIONS@ returns a boolean in its array of results indicating whether a driver supports an ODBC function. This function is implemented in the Driver Manager. It can also be implemented in drivers. If a driver implements ODBC_GET_FUNCTIONS@, the Driver Manager calls the funct6ion in the driver. Otherwise, it implements the function itself.

The func argument can be one of the following values:

ODBC Core Functions

SQL_API_SQLALLOCCONNECT SQL_API_SQLFETCH
SQL_API_SQLALLOCENV SQL_API_SQLFREE_CONNECT
SQL_API_SQLALLOCSTMT SQL_API_SQLFREEENV
SQL_API_SQLBINDCOL SQL_API_SQLFREESTMT
SQL_API_SQLCANCEL SQL_API_SQLGETCURSORNAME
SQL_API_SQLCOLATTRIBUTES
SQL_API_SQLNUMRESULTSCOL
SQL_API_SQLCONNECT SQL_API_SQLPREPARE
SQL_API_SQLDESCRIBECOL SQL_API_SQLROWCOUNT
SQL_API_SQLDISCONNECT SQL_API_SQLSETCURSORNAME
SQL_API_SQLERROR SQL_API_SQLSETPARAM
SQL_API_SQLEXECDIRECT SQL_API_SQLTRANSACT
SQL_API_SQLEXECUTE

ODBC Extension Level 1 Functions

SQL_API_SQLBINDPARAMETER
SQL_API_SQLGETTYPEINFO SQL_API_SQLCOLUMNS
SQL_API_SQLPARAMDATA SQL_API_SQLDRIVERCONNECT
SQL_API_SQLPUTDATA SQL_API_SQLGETCONNECTOPTION
SQL_API_SQLSETCONNECTOPTION
SQL_API_SQLGETDATA SQL_API_SQLSETSTMTOPTION
SQL_API_SQLGETFUNCTIONS
SQL_API_SQLSPECIALCOLUMNS

SQL_API_SQLGETINFO SQL_API_SQLSTATISTICS
SQL_API_SQLGETSTMTOPTION
SQL_API_SQLTABLES

ODBC Extension Level 2 Functions

SQL_API_SQLBROWSECONNECT SQL_API_SQLNUMPARAMS
SQL_API_SQLCOLUMNPRIVILEGES SQL_API_SQLPARAMOPTIONS
SQL_API_SQLDATASOURCES SQL_API_SQLPRIMARYKEYS
SQL_API_SQLDESCRIBEPARAM
SQL_API_SQLPROCEDURECOLUMNS
SQL_API_SQLDRIVERS SQL_API_SQLPROCEDURES
SQL_API_SQLEXTENDEDFETCH SQL_API_SQLSETPOS
SQL_API_SQLFOREIGNKEYS
SQL_API_SQLSETSCROLLOPTIONS
SQL_API_SQLMORERESULTS
SQL_API_SQLTABLEPRIVILEGES
SQL_API_SQLNATIVESQL

See also ODBC_GET_INFO@

ODBC_GET_STMT_OPTION@

Set options related to a statement handle

Format ODBC_GET_STMT_OPTION@(format sql_channel@ channel, stmtHandle, foption)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

option — Option to set

param — Value associated with the option.

Description Returns the value of a statement option. The foption parameter must be one of the following:

SQL_ASYNC_ENABLE	SQL_BIND_TYPE
SQL_CONCURRENCY	SQL_CURSOR_TYPE
SQL_KEYSET_SIZE	SQL_MAX_LENGTH
SQL_MAX_ROWS	SQL_NOSCAN
SQL_QUERY_TIMEOUT	SQL_RETRIEVE_DATA
SQL_ROWSET_SIZE	SQL_SIMULATE_CURSOR

SQL_ROW_NUMBER - returns a 32-bit integer value that specifies the number of the current row in the entire result set. If the number cannot be determined, or there is no current row, the driver returns 0.

See also ODBC_SET_STMT_OPTION@

ODBC_GET_TYPE_INFO@

Returns the data types supported by a data source

Format ODBC_GET_TYPE_INFO@(format sql_channel@ channel, dbHandle, infoType)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

dbHandle — A connection handle as returned by ODBC_ALLOC_CONNECT@

infoType — An SQL data type

Description ODBC_GET_TYPE_INFO@ returns the results as a standard result set. The columns in the result set are as follows:

- **TYPE_NAME** (string) - the data-source dependent data type name. For example, "CHAR", "VARCHAR", "MONEY", "LONG VARBINARY", or "CHAR() FOR BIT DATA". Applications must use this name in **CREATE TABLE** and **ALTER TABLE** statements.
- **DATA_TYPE** (string) - the SQL data type. This can be an ODBC SQL data type or a driver-specific SQL data type.

- **PRECISION** (Integer) - The maximum precision of the data type on the data source. NULL is returned for data types where precision is not applicable.
- **LITERAL_PREFIX** (String) - Characters used to prefix a literal. For example, a single-quote (') for character data types or 0x for binary data types. NULL is returned for data types where a literal prefix is not applicable.
- **LITERAL_SUFFIX** (String) - Characters used to terminate a literal. For example, a single-quote (') for character data types; NULL is returned for data types where a literal suffix is not applicable.
- **CREATE_PARAMS** (string) - parameters for a data definition. For example, CREATE_PARAMS for DECIMAL would be "precision, scale"; CREATE _PARAMS for VARCHAR would equal "max length"; NULL is returned if there are no parameters for the data type definition, for example INTEGER.
- **NULLABLE** (Integer) - determines whether the data type accepts a NULL value. SQL_NO_NULLS if the data type does not accept NULL values. SQL_NULLABLE if the data type accepts NULL values.
- **CASE_SENSITIVE** - whether a character data type is case sensitive in collations and comparisons. TRUE if the data type is a character data type and is case-sensitive. FALSE if the data type is not a character data type and is not case-sensitive.
- **SEARCHABLE** - How the data type is used in a WHERE clause. SQL_LIKE_ONLY if the data type can be used in a **WHERE** clause only with the like predicate. SQL_ALL_EXCEPT_LIKE if the data type can be used in a **WHERE** clause with all comparison operators except **LIKE**. SQL_SEARCHABLE if the data type can be used in a **WHERE** clause with any comparison operator.
- **UNSIGNED_ATTRIBUTE** - TRUE if the data type is unsigned, FALSE if the ata type is signed. NULL is returned if the attribute is not applicable to the data type or the data type is not numeric.
- **MONEY** - TRUE if the data type is a money data type. FALSE if it is not.
- **AUTO_INCREMENT** - TRUE if the data type is autoincrementing. FALSE if the data type is not autoincrementing. NULL is returned if the attribute is not applicable to the data type or the data type is not numeric. An application can insert values into a column having this arrtibute, but cannot update the values in this column.
- **LOCAL_TYPE_NAME** (Integer) - The localized version of the data-source dependent name of the data type. NULL is returned if a localized name is not supported by the data source. This name is intended for display only, such as in dialog boxes.
- **MINIMUM SCALE** (Integer) - The minimum scale of the data type on the data source. If a data type has a fixed scale, the MINIMUM_SCALE and MAXIMUM_SCALE columns both contain this value. For example, an SQL_TIMESTAMP column might have a fixed scale for fractional seconds. NULL is returned where the scale is not applicable.
- **MAXIMUM SCALE** (Integer) - The maximum scale of the data type on the data source. NULL is returned where the scale is not applicable. If the maximum scale is not defined separately on the

data source, but is instead defined to be the same as the maximum precision, this column contains the same value as the precision column.

Attribute information can apply to data types or to specific columns in a result set. ODBC_GET_TYPE_INFO@ returns information about attributes associated with data types. ODBC_COL_ATTRIBUTES@ returns information about attributes associated with columns in a result set.

See also ODBC_BIND_COL@

ODBC_MORE_RESULTS@

Determines if more results are pending on a statement handle

Format ODBC_MORE_RESULTS@(format sql_channel@ channel, stmtHandle)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

Description **SELECT** statements return result sets. **UPDATE**, **INSERT**, and **DELETE** statements return a count of affected rows. If any of these statements are batched, submitted with arrays of parameters, or in procedures, they can return multiple result sets or counts.

If another result set or count is available, ODBC_MORE_RESULTS@ returns SQL_SUCCESS and initializes the result set or count for additional processing. After calling ODBC_MORE_RESULTS@ for Select statements, an application can call functions to determine the characteristics of the result set. After calling ODBC_MORE_RESULTS@ for **UPDATE**, **INSERT**, or **DELETE** statements, an application can call **ODBC_ROW_COUNT@**.

If all results have been processed, ODBC_MORE_RESULTS@ returns SQL_NO_DATA_FOUND. Note that if there is a current result set with unfetched rows, ODBC_MORE_RESULTS@ discards the result set and makes the next result set or count available.

If a batch of statements or a procedure mixes other SQL statements with **SELECT, UPDATE**, **INSERT**, and **DELETE** statements, these statements do not affect ODBC_MORE_RESULTS@.

See also ODBC_FETCH@

ODBC_NATIVE_SQL@

Returns the SQL string as translated by the driver

Format ODBC_NATIVE_SQL@(format sql_channel@ channel, dbHandle, sqlString)

Arguments

channel	an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:	
	host	A string containing the name of the ODBC gateway
	id	An Applixware task ID as returned by ODBC_START_SERVER@
dbHandle	A connection handle as returned by ODBC_ALLOC_CONNECT@	
sqlString	SQL text string to be translated	

Description Returns an array containing three elements, as follows:

- return[0] is the return code
- return[1] contains the translated string if the Macro is successful.
- return[2] is error information

The sql_channel@ format is defined in dbase_.am. It contains the following information:

format sql_channel@

host,	A string containing the name of the ODBC gateway
id	An Applixware task ID as returned by ODBC_START_SERVER@

The following are examples of what ODBC_NATIVE_SQL@ might return for the following input SQL string containing the scalar function CONVERT. Assume that the column empid is of type INTEGER in the data source:

SELECT { fn CONVERT (empid, SQL_SMALLINT) } FROM employee

A driver for SQL Server might return the following translated SQL string:

SELECT convert (smallint, empid) FROM employee

A driver for ORACLE Server might return the following translated SQL string:

SELECT to_number (empid) FROM employee

See also ODBC_START_SERVER@, ODBC_ALLOC_CONNECT@, ODBC_PREPARE@

ODBC_NUM_PARAMS@

Returns the number of parameters in an SQL statement

Format ODBC_NUM_PARAMS@(format sql_channel@ channel, stmtHandle)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

Description ODBC_NUM_PARAMS@ can only be called after ODBC_PREPARE@ has been called.

ODBC_NUM_PARAMS@ returns an array of 3 elements, as follows:

- return[0] is the return code
- return[1] contains the number of parameters in the SQL statement.
- return[2] is error information

If the statement associated with stmtHandle does not contain parameters, ODBC_NUM_PARAMS@ returns zero in the return array.

See also ODBC_PREPARE@

ODBC_NUM_RESULT_COLS@

Returns the number of columns in a result set

Format ODBC_NUM_RESULT_COLS@(format sql_channel@ channel, stmtHandle)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway

id An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle An SQL statement handle as returned by ODBC_ALLOC_STMT@.

Description ODBC_NUM_RESULT_COLS@ can be successfully called only when the stmtHandle is in the prepared, executed, or positioned state.

The sql_channel@ format is defined in dbase_.am. It contains the following information:

format sql_channel@

host, A string containing the name of the ODBC gateway

id An Applixware task ID as returned by ODBC_START_SERVER@

ODBC_NUM_RESULT_COLS@ returns an array of 3 elements, as follows:

- return[0] is the return code
- return[1] is an array of results. If the macro is successful, this array contains the number of rows of data in the result set referenced by the stmtHandle.
- return[2] is error information

See also ODBC_ALLOC_STMT@

ODBC_PARAM_DATA@

Supplies parameter data at statement execution time

Format ODBC_PARAM_DATA@(format sql_channel@ channel, stmtHandle)

Arguments channel an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

host A string containing the name of the ODBC gateway

id An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle An SQL statement handle as returned by ODBC_ALLOC_STMT@.

Description This macro is used in conjunction with ODBC_PUT_DATA@ to supply parameter data at statement execution time. See ODBC_BIND_PARAMETER_DATA_AT_EXEC@ for more information.

ODBC_PREPARE@

Prepares an SQL statement for execution

Format ODBC_PREPARE@(format sql_channel@ channel, stmtHandle, sqlString)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

sqlString — SQL text string to be prepared

Description An application calls ODBC_PREPARE@ to send an SQL statement to the data source for preparation. The application can include one or more parameter markers in the SQL statement. To include a parameter marker, the application embed a question mark (?) into the SQL statement at the appropriate location.

Note that if an application uses ODBC_PREPARE@ to prepare and ODBC_EXECUTE@ to submit a COMMIT or ROLLBACK statement, the application will not be interoperable between DBMS products. To commit or roll back a transaction, call ODBC_TRANSACT@.

The driver modifies the statement to use the for of SQL recognized by the data source, then submits it to the data source for preparation. In particular, the driver modifies the escape clauses used to define ODBC-specific SQL. For the driver, a stmtHandle is similar to a statement identifier in embedded SQL code. If the data source supports statement identifiers, the driver can send a statement identifier and parameter values to the data source.

Once a statement is prepared, the application uses stmtHandle to refer to the statement in later function calls. The prepared statement associated with the stmtHandle may be reexecuted by calling ODBC_EXECUTE@ until the application frees the statement handle with a call to ODBC_FREE_STMT@ with the SQL_DROP option or until the stmtHandle is used in a call to ODBC_PREPARE@, ODBC_DIRECT@, or one of the catelog functions (ODBC_COLUMNS, ODBC_TABLES, and so on). Once the application prepares a statement, it can request information about the format of the result set.

Some drivers cannot return syntax errors or access violations when the application calls ODBC_PREPARE@. A driver may handle syntax errors and access violations, only syntax errors, or neither. Therefore, an application must be able to handle these conditions when calling subsequent related functions such as

ODBC_NUM_RESULT_COLS@, ODBC_DESCRIBE_COL@, ODBC_COL_ATTRIBUTES@, and ODBC_EXECUTE@.

Depending on the capabilities of the driver and data source and on whenther the application has called ODBC_BIND_PARAMETER@, parameter information (such as data types) might be checked when the statement is prepared or when it is executed. For maximum interoperability, an application should unbind all the parameters that applied to an old SQL statement before preparing a new SQL statement on the same stmtHandle. This prevents errors that are due to old parameter information being applied to the new statement.

It is important to remember that committing or rolling back a transaction, either by calling ODBC_TRANSACT@ or by using the SQL_AUTOCOMMIT connection option, can cause the data source to delete the access plans for all statement handles associated with a database handle. For more information, see the SQL_CURSOR_COMMIT_BEHAVIOR and SQL_CURSOR_ROLLBACK_BEHAVIOR information types in ODBC_GET_INFO@.

See also ODBC_GET_INFO@, ODBC_ALLOC_STMT@

ODBC_PRIMARY_KEYS@

Returns the column names that comprise the primary key for a table

Format ODBC_PRIMARY_KEYS@(format sql_channel@ channel, stmtHandle, tableQualifier, tableOwner, tableName)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

tableQualifier — Table qualifier. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty empty string ("") denotes those tables that do not have qualifiers.

tableOwner — Table owner. If a driver supports owners for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty empty string ("") denotes those tables that do not have owners.

tableName — Table name.

Description Returns a standard result set, ordered by TABLE_QUALIFIER, TABLE_OWNER, TABLE_NAME, and KEY_SEQ. The result set contains six columns, all of which are variable-length strings with a 128-byte maimum length unless otherwise noted. The columns are as follows:

- **TABLE_QUALIFIER** is a Primary key table qualifier identifier. This is NULL if not applicable to the data source. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty empty string ("") denotes those tables that do not have qualifiers.
- **TABLE_OWNER** is a Primary key table owner identifier. This is NULL if not applicable to the data source. If a driver supports owners for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty empty string ("") denotes those tables that do not have owners.
- **TABLE_NAME** is a Primary key table identifier.
- **COLUMN_NAME** is a Primary key column identifier.
- **KEY_SEQ** is the column sequence number in key, starting with 1. This data is an integer value.
- **PK_NAME** is a Primary key identifier. This is NULL if not applicable to the data source. This column was added in ODBC 2.0. ODBC 1.0 drivers may return a different, driver-specific column with the same column number.

To retrieve information in a result set, use ODBC_BIND_COL@ and ODBC_FETCH@.

See also ODBC_FETCH@

ODBC_PROCEDURES@

Returns a list of procedure names

Format ODBC_PROCEDURES@(format sql_channel@ channel, stmtHandle, procQualifier, procOwner, procName)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

procQualifier Procedure qualifier. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty empty string ("") denotes those tables that do not have qualifiers.

procOwner String search pattern for procedure owner names. If a driver supports owners for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty empty string ("") denotes those tables that do not have owners.

procName String search pattern for procedure names.

Description A procedure is an executable object, or a named entity, that can be called using input and output parameters. A procedure can return result sets similar to the results returned by SQL **SELECT** expressions.

ODBC_PROCEDURES@ lists all procedures in the requested range. A user may or may not have permissions to execute any of these procedures. To check accessibility, an application can call ODBC_GET_INFO@ and check the SQL_ACCESSIBLE_PROCEDURES information value. Otherwise, the application must be able to handle a situation where the user selecta procedure which it cannot execute.

ODBC_PROCEDURES@ might not return all procedures. Applications can use any valid procedure, regardless of whether it is returned by ODBC_PROCEDURES@.

ODBC_PROCEDURES@ returns a standard result set, ordered by PROCEDURE_QUALIFIER, PROCEDURE_OWNER, and PROCEDURE_NAME. The result set contains eight columns, all of which are variable-length strings with a 128-byte maimum length unless otherwise noted. The columns are as follows:

- **PROCEDURE_QUALIFIER** is a procedure qualifier identifier. This is NULL if not applicable to the data source. If a driver supports qualifiers for some procedures but not for others, such as when a driver retrieves data from different DBMSs, an empty empty string ("") denotes those procedures that do not have qualifiers.
- **PROCEDURE_OWNER** is a procedure owner identifier. This is NULL if not applicable to the data source. If a driver supports owners for some procedures but not for others, such as when a driver retrieves data from different DBMSs, an empty string ("") denotes those procedures that do not have owners.
- **PROCEDURE_NAME** is a procedure name.
- **NUM_INPUT_PARAMS** reserved for future use.
- **NUM_OUTPUT_PARAMS** reserved for future use.
- **NUM_RESULT_SETS** reserved for future use.
- **REMARKS** describe the procedure. This is a variable-length string of up to 254 bytes.
- **PROCEDURE_TYPE** defines the procedure type. SQL_PT_UNKNOWN means that it cannot be determined whether the procedure returns a value. SQL_PT_PROCEDURE means that the

object is a procedure, and does not return a value. SQL_PT_FUNCTION means that the object is a function, and returns a value.

See also ODBC_FETCH@, ODBC_GET_INFO@, ODBC_CANCEL@

ODBC_PROCEDURE_COLUMNS@

Returns a list of procedure parameters

Format ODBC_PROCEDURE_COLUMNS@(format sql_channel@ channel, stmtHandle, procQualifier, procOwner, procName, colName)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

procQualifier — Procedure qualifier. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty empty string ("") denotes those tables that do not have qualifiers.

procOwner — String search pattern for procedure owner names. If a driver supports owners for some tables but not for others, such as when a driver retrieves data from different DBMSs, an empty empty string ("") denotes those tables that do not have owners.

procName — String search pattern for procedure names.

colName — String search pattern for column names.

Description A procedure is an executable object, or a named entity, that can be called using input and output parameters. A procedure can return result sets similar to the results returned by SQL **SELECT** expressions.

ODBC_PROCEDURES@ returns a standard result set, ordered by PROCEDURE_QUALIFIER, PROCEDURE_OWNER, and PROCEDURE_NAME. The result set contains eight columns, all of which are variable-length strings with a 128-byte maimum length unless otherwise noted. The columns are as follows:

- **PROCEDURE_QUALIFIER** is a procedure qualifier identifier. This is NULL if not applicable to the data source. If a driver supports qualifiers for some procedures but not for others, such as when a

driver retrieves data from different DBMSs, an empty empty string ("") denotes those procedures that do not have qualifiers.

- **PROCEDURE_OWNER** is a procedure owner identifier. This is NULL if not applicable to the data source. If a driver supports owners for some procedures but not for others, such as when a driver retrieves data from different DBMSs, an empty string ("") denotes those procedures that do not have owners.
- **PROCEDURE_NAME** is a procedure name.
- **COLUMN_NAME** is a column identifier.
- **DATA_TYPE** is a SQL data type. This can be an ODBC SQL data type or a driver-specific SQL data type.
- **TYPE_NAME** is a data-source dependent data type name; for example, "CHAR", "VARCHAR", "MONEY", "LONG VARBINARY", or "CHAR () FOR BIT DATA".
- **PRECISION** is the precision of a column on the data source. NULL is returned where precision is not applicable.
- **LENGTH** is the length, in bytes, of data transferred on an ODBC_FETCH@ or ODBC_GET_DATA@ operation if SQL_C_DEFAULT is specified. For numeric data, this size may be different from the size of the data stored on the data source. This value is the same as the **PRECISION** column for character or binary data.
- **SCALE** is the scale of the column on the data source. NULL is returned for data types where scale is not applicable.
- **RADIX** is for numeric data types. This field is either "10" or "2". **10** indicates that the values in PRECISION and SCALE give the number of decimal digits allowed for the column. For example, a **DECIMAL**(12,5) column would return a **RADIX** of 10, a **PRECISION** of 12, and a **SCALE** of 5; A **FLOAT** column could return a **RADIX** of 10, a **PRECISION** of 15, and a **SCALE** of NULL.

  2 indicates that the values in **PRECISION** and **SCALE** give the number of bits allowed in the column. For example, a **FLOAT** column could return a **RADIX** of 2, a **PRECISION** of 52, and a **SCALE** of NULL.

  NULL is returned for all data types where radix is not applicable.
- **NULLABLE** is one of the following values:

  SQL_NO_NULLS means the column does not accept NULL values.

  SQL_NULLABLE meams the column accepts NULL values.

  SQL_NULLABLE_UNKNOWN meams the column accepts NULL values.
- **REMARKS** contains a text description of the column.

Description ODBC_PROCEDURES@

ODBC_PUT_DATA@

Sends data for a parameter or column at statement execution time

Format ODBC_PUT_DATA@(format sql_channel@ channel, stmtHandle, data)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

data — Data for the parameter marker or column.

Description For an explanation of how data-at-execution parameter data is passed at statement execution time, see **ODBC_BIND_PARAMETER@**. For an explanation of how data-at-execution is updated or added, see ODBC_SET_POS@.

An application can use ODBC_PUT_DATA@ to send data in parts only when sending either character C data or binary C data to a column with a character, binary, or data-source-specific data type. If ODBC_PUT_DATA@ is called more than once under any other conditions, it returns SQL_ERROR and (Numeric value out of range.)

See also **ODBC_CANCEL@**, **ODBC_PARAM_DATA@**, **ODBC_EXECUTE@**

ODBC_ROW_COUNT@

Returns the number of rows affected by a SQL statement

Format ODBC_ROW_COUNT@(format sql_channel@ channel, stmtHandle)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle An SQL statement handle as returned by ODBC_ALLOC_STMT@.

Description Return the number of rows affected by an **UPDATE**, **INSERT**, or **DELETE** statement or by a SQL_UPDATE, SQL_ADD, or SQL_DELETE operation in ODBC_SET_POS@.

Description ODBC_EXECUTE@, ODBC_EXECUTE_DIRECT@

ODBC_SET_CONNECT_OPTION@

Sets the parameters for an ODBC connection

Format ODBC_SET_CONNECT_OPTION@(format sql_channel@ channel, dbHandle, Option, value)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

dbHandle — A connection handle as returned by ODBC_ALLOC_CONNECT@

Option — A constant value indicating the connection option to set

Value — The value to set the connection option to

Description If successful, this macro returns either SQL_SUCCESS or SQL_SUCCESS_WITH_INFO. These constants are defined in dbase_.am.

ODBC_SET_CONNECT_OPTION@ established parameters for a connection to a database.

The connection options that can be changed are as follows:

SQL_ACCESS_MODE determines whether the connection is read-only, read/write, and so on. The possible values are SQL_MODE_READ_WRITE (the default), and SQL_MODE_READ_ONLY.

SQL_AUTOCOMMIT determines whether to use auto-commit or manual commit mode. SQL_AUTOCOMMIT_OFF establishes that the application must explicitly commit or roll back transaction with ODBC_TRANSACT@. SQL_AUTOCOMMIT_ON is the default. This means that each SQL statement is committed immediately after it is executed.

SQL_LOGIN_TIMEOUT determines the number of seconds to wait for a login request to complete before returning to the application. If this is set to zero (0), the connection attempt will wait indefinately.

SQL_OPT_TRACE tells the Driver Manager whether to use tracing. If the parameter is SQL_OPT_TRACE_OFF, no tracing is enabled. If the parameter is SQL_OPT_TRACE_ON, trace information is written to the trace file established with the SQL_OPT_TRACEFILE option. If this option is not specified, the driver writes trace information to the file sql.log in the current directory.

SQL_OPT_TRACEFILE establishes the file into which trace information is written. The parameter is a string containing the pathname of the trace file.

SQL_TXN_ISOLATION sets the transaction isolation level for the current connection. An application must call ODBC_TRANSACT@ to commit or roll back all open transactions on the connection before calling ODBC_SET_CONNECT_OPTION@ with this option. The following discussion uses three terms that require clarification: *dirty reads*, *nonrepeatable reads*, and *phantoms*.

Dirty Read

Transaction 1 changes a row. Transaction 2 reads the changed row before transaction 1 commits the change. If transaction 1 rolls back the change, transaction 2 will have read a row that is considered to have never existed.

Nonrepeatable Read

Transaction 1 reads a row. Transaction 2 updates or deletes this row and commits the change. If Transaction 1 attempts to reread the row, it will receive different row values or discover that the row has been deleted.

Phantom

Transaction 1 reads a set of rows that satisfy some search criteria. Transaction 2 inserts a row that matches the search criteria. If transaction 1 reexecutes the statement that read the rows, it receives a different set of rows.

The possible values for SQL_TXN_ISOLATION are:

- SQL_TXN_READ_UNCOMMITTED means dirty reads, nonrepeatable reads, and phantoms are possible.
- SQL_TXN_READ_COMMITTED means dirty reads are not possible. Nonrepeatable reads and phantoms are possible.
- SQL_TXN_REPEATABLE_READ means dirty reads and nonrepeatable reads are not possible. Phantoms are possible.
- SQL_TXN_SERIALIZABLE means transactions are serialized. Dirty reads, nonrepeatable reads and phantoms are not possible.
- SQL_TXN_VERSIONING means transactions are serializable, but higher concurrency is possible that with SQL_TXN_SERIALIZABLE. Dirty reads are not possible. Typically,

SQL_TXN_SERIALIZABLE is implemented using locking protocols that reduce concurrency and SQL_TXN_VERSIONING is implemented using a non-locking protocol such as record versioning.

SQL_CURRENT_QUALIFIER establishes a qualifier to be used by the data source. This qualifier could be the name of a database, or a directory. Valid qualifiers are database-specific.

SQL_ODBC_CURSORS specifies how the Driver Manager uses the ODBC cursor library. Possible values are as follows:

SQL_CUR_USE_IF_NEEDED means the Driver Manager uses the ODBC cursor library only if it is needed. If the driver supports the SQL_FETCH_PRIOR option in the ODBC_EXTENDED_FETCH@ macro, the Driver Manager uses the scrolling capabilities of the driver. Otherwise, it uses the ODBC cursor library.

SQL_CUR_USE_ODBC means the driver uses the ODBC cursor library.

SQL_CUR_USE_DRIVER means the Driver Manager uses the scrolling capablities of the driver. This is the default.

SQL_PACKET_SIZE specifies the network packet size in bytes. Many data sources either do not support this option or can only return the network packet size. If the specified size exceeds the maximum packet size or is smaller than the minimum packet size, the driver substitutes either the maximum or minimum value and returns SQLSTATE 01s02 (option value changed).

See also ODBC_GET_CONNECT_OPTION@

ODBC_SET_CURSOR_NAME@

Associates a cursor name with an active statement handle

Format ODBC_SET_CURSOR_NAME@(format sql_channel@ channel, stmtHandle, cursorName)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

cursorName — Name of the cursor.

Description The only ODBC SQL statements that use a cursor name are a positioned update and delete. For example:

UPDATE *table-name* ... WHERE CURRENT OF *cursor-name*

If the application does not call ODBC_SET_CURSOR_NAME@ to define a cursor name, on execution of a **SELECT** statement the driver generates a cursor name that begins with the letters SQL_CUR and does not exceed 18 characters in length.

All cursor names associated with a database handle must be unique. The maximum length of a cursor name is defined by the driver. For maximum interoperability, it is recommended that applications limit cursor names to no more than 18 characters.

A cursor name that is set either explicitly or implicitly remains set until the statement handle with which it is associated is dropped. Use ODBC_FREE_STMT@ with the SQL_DROP option to free a statement handle.

See also ODBC_GET_CURSOR_NAME@

ODBC_SET_POS@

Sets the cursor position in a rowset

Format ODBC_SET_POS@(format sql_channel@ channel, stmtHandle, iRow, fOption, fLock)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

iRow — The position of the row in the rowset on which to perform the operation specified with the fOption argument. If irow is 0, the operation applies to every row in the rowset.

fOption — Option to perform:

SQL_POSITION

SQL_REFRESH

SQL_DELETE

The options SQL_UPDATE and SQL_ADD are not supported.

fLock — Specifies how to lock the row after the operation indicated by fOption

Description ODBC_SET_POS@ sets the position of a cursor in the rowset, and allows an application to refresh and delete rows from the row set.

The foption argument specifies the operation to perform on the row set. To determine which options are supported by a data source, the application calls ODBC_GET_INFO@ with the SQL_POS_OPERATIONS information type.

SQL_POSITION - positions the cursor on the row specified by irow.

SQL_REFRESH - positions the cursor on the row specified by irow, and refreshes the data in the rowset buffers for that row. For more information about how the driver returns data in the rowset buffers, see the descriptions of column-wise binding in ODBC_EXTENDED_FETCH@.

SQL_DELETE positions the cursor on the row specified by irow, and deletes the underlying row of data. It changes the RowStatus array specified in ODBC_EXTENDED_FETCH@ to SQL_ROW_DELETED. After the row has been deleted, calls to ODBC_GET_DATA@ and calls to ODBC_SET_POS@ with fOption set to anything except SQL_POSITION are not valid for the row.

Whether the row remains visible depends on the cursor type. For example, deleted rows are visible to static and keyset-driven cursors but invisible to dynamic cursors.

The fLock argument provides a way for applications to control concurrency and simulate transactions on data sources that do not support them. Generally, data sources that support concurrency levels and transactions will only support the SQL_LOCK_NO_CHANGE value of the fLock argument.

The fLock argument specifies the lock state of the row after ODBC_SET_POS@ has been executed. To stimulate a transaction, an application uses the SQL_LOCK_RECORD macro to lock each of the rows in the transaction. It then uses the SQL_UPDATE_RECORD or SQL_DELETE_RECORD macro to update or delete each row; the driver may temporarily change the lock state of the row while performing the operation specified by the fOption argument. Finally, it uses the SQL_UNLOCK_RECORD macro to unlock each row. If the driver is unable to lock the row either to perform the rquested operation or to satisfy the fLock argument, it returns SQL_ERROR and SQLSTATE 42000.

Although the fLock argument is specified for a statement handle, the lock accords the same privileges to all statement handles on the connection. In particular, a lock that is acquired by one statement handle on a connection can be unlocked by a different statement handle on the same connection.

A row locked through ODBC_SET_POS@ remains locked until the application calls ODBC_SET_POS@ for the row with fLock set to SQL_LOCK_UNLOCK or the applications calls ODBC_FREE_STMT@ with the SQL_CLOSE or SQL_DROP option.

The fLock argument supports the following types of locks. To determine which locks are supported by a data source, an application calls ODBC_GET_INFO@ with the SQL_LOCK_TYPES information type.

SQL_LOCK_NO_CHANGE means that the driver or data source ensures that the row is in the same locked or unlocked state it was before ODBC_SET_POS@ was called.

This value of fLock allows data sources that do not support explicit low-level locking to use whatever locking is required by the current concurrency and transaction isolation levels.

SQL_LOCK_EXCLUSIVE means the driver or data source locks the row exclusively. A statement handle on a different database handle or in a different application cannot be used to acquire any locks on the row.

SQL_LOCK_UNLOCK means the driver or data source unlocks the row.

For the delete operation, ODBC_SET_POS@ uses the fLock argument as follows:

- If the application sets fLock to SQL_LOCK_NO_CHANGE, the driver guarantees that a delete operation will succeed only if the application specified SQL_CONCUR_LOCK for the SQL_CONCURRENCY statement option.
- If the application specifies SQL_CONCUR_READ_ONLY for the SQL_CONCURRENCY statement option, the driver rejects any delete operation.

Performing Bulk Operations

If the iRow argument is 0, the driver performs the operation specified in the fOption argument for every row in the rowset. If an error occurs that pertains to the entire rowset, such as SQLSTATE S1T00 (Timeout Expired), the driver returns SQL_ERROR and the appropriate SQLSTATE. The contents f the rowset buffers are undefined, and the cursor position is unchanged.

If an error occurs that pertains to a single row, the driver:

- Sets the element in the rgfRowStatus array for the row to SQL_ROW_ERROR.
- Posts SQLSTATE 01S01 (Error in row) in the error queue.
- Posts one or more additional SQLSTATEs for the error after SQLSTATE 01S01 (Error in row) followed by zero or more additional SQLSTATEs.
- After it has processed the error or warning, the driver continues the operation for the remaining rows in the rowset and returns SQL_SUCCESS_WITH_INFO Thus, for each row that returned an error, the error queue contains SQLSTATE 01S01 (Error in row) followed by zero or more additional SQLSTATEs.
- If the driver returns any warnings, such as SQLSTATE 01004 (Data truncated), it returns warnings that apply to the entire rowset or to unknown rows in the rowset before it returns the error information that applies to specific rows. It returns warnings for specific rows along with any error information about those rows.

See also ODBC_EXTENDED_FETCH@, ODBC_BIND_COL@

ODBC_SET_STMT_OPTION@

Set options related to a statement handle

Format ODBC_SET_STMT_OPTION@(format sql_channel@ channel, stmtHandle, foption, vparam)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

option — Option to set

param — Value associated with the option.

Description Statement options for a statement handle remain in effect until they are changed by another call to ODBC_SET_STMT_OPTION@ or the statement handle is dropped using ODBC_FREE_STMT@ with the SQL_DROP option. Calling ODBC_FREE_STMT@ with the SQL_CLOSE, SQL_UNBIND, or SQL_RESET_PARAMS options does not reset statement options.

Some statement options support substitution of a similar value if the data source does not support the specified value of param. In such cases, the driver returns SQL_SUCCESS_WITH_INFO and SQLSTATE 01S02 (option value changed). For example, if foption is SQL_CONCURRENCY, and param is SQL_CONCUR_ROWVER, and the data source does not support this, the driver substitutes SQL_CONCUR_VALUES. To determine the substituted value, the application must call ODBC_GET_STMT_OPTION@.

The currently defined option are listed below. The format of the information set with param depends on the specified option. ODBC_SET_STMT_OPTION@ accept option information as a string or a 32-bit integer value. The format for each is noted in the description of each option. The format applies to the information returned for each option in ODBC_GET_STMT_OPTION@.

SQL_ASYNC_ENABLE

SQL_BIND_TYPE

SQL_CURSOR_TYPE

SQL_KEYSET_SIZE

SQL_MAX_LENGTH
SQL_MAX_ROWS
SQL_NOSCAN
SQL_QUERY_TIMEOUT
SQL_RETRIEVE_DATA
SQL_ROWSET_SIZE
SQL_SIMULATE_CURSOR
SQL_USE_BOOKMARKS

See also ODBC_GET_STMT_OPTION@

ODBC_SPECIAL_COLUMNS@

Retrieves information about columns in a table

Format ODBC_SPECIAL_COLUMNS@(format sql_channel@ channel, stmtHandle, colType, tableQualifier, tableOwner, tableName, tableType, Scope, Nullable)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

colType — Type of column to return. Must be one of the following values:

SQL_BEST_ROWID - Returns the optimal column or set of columns that, by retrieving values from the column or columns, allows any row in the specified table to be uniquely identified. A column can be either a pseudocolumn specifically designed for this purpose (as in Oracle ROWID) or the column or columns of any unique index for the table.

SQL_BEST_ROWVER - Returns the column or columns in the specified table, if any, that are automatically updated by the data source when any value in the row is updated by any transaction (as in SQLBase ROWID or Sybase TIMESTAMP).

tableQualifier Qualifier name. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, and empty empty string ("") denotes those tables that do not have qualifiers.

tableOwner String search pattern for owner names.

tableName String search pattern for table names. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, and empty empty string ("") denotes those tables that do not have owners

tableType List of table types to match.

scope Minimum required scope of the rowid. The returned rowid may be of greater scope. Must be one of the following:

SQL_SCOPE_CURROW - The rowid is guaranteed to be valid only while positioaned on that row. A later reselect using rowid may not return a rowif the row was updated or deleted by another transaction.

SQL_SCOPE_TRANSACTION - The rowid is guaranteed to be valid for the duration of the current process.

SQL_SCOPE_SESSION - The rowid is guaranteed to be valid for the duration of the session (across transaction boundaries).

nullable Determines whether to return special columns that have a NULL value. Must be one of the following:

SQL_NO_NULLS - exclude special columns that have NULL values.

SQL_NULLABLE - return special columns even if they have NULL values.

Description ODBC_SPECIAL_COLUMNS@ is provided so that applications can provide their own custom scrollable-cursor functionality, similar to that provided by ODBC_EXTENDED_FETCH@ and ODBC_STMT_OPTION@.

When the colType argument is SQL_BEST_ROWID, ODBC_SPECIAL_COLUMNS@ returns the column or columns that uniquely identify each row in the table. These columns can always be used in a *select-list* or **WHERE** clause. However, ODBC_COLUMNS@ does not necessarily return these columns. For example, ODBC_COLUMNS@ might not return the Oracle ROWID pseudo-column ROWID. If there are no columns that uniquely identify each row in the table, ODBC_SPECIAL_COLUMNS@ returns a rowset with no rows; a subsequent call to ODBC_FETCH@ or ODBC_EXTENDED_FETCH@ on the statment handle returns SQL_NO_DATA_FOUND.

If the colType, scope or nullable arguments specify characteristics that are not supported by the data source, ODBC_SPECIAL_COLUMNS@ returns a result set with no rows. (The developer might ordinarily expect the macro to return SQL_ERROR with SQLSTATE S1C00 (Driver Not Capable). This does not happen.) A subsequent call to

ODBC_FETCH@ or ODBC_EXTENDED_FETCH@ on the statement handle returns SQL_NO_DATA_FOUND.

ODBC_SPECIAL_COLUMNS@ returns the results as a standard result set, ordered by SCOPE. The following table lists the columns in the result set.

- **SCOPE** (Integer) - Scope of the rowid. Contains one of the following values: SQL_SCOPE_CURROW, SQL_SCOPE_TRANSACTION, SQL_SCOPE_SESSION

  NULL is returned when colType is SQL_ROWVER.

- **DATA_TYPE** (string) - the SQL data type. This can be an ODBC SQL data type or a driver-specific SQL data type.
- **TYPE_NAME** (string) - the data-source dependent data type name; for example, "CHAR", "VARCHAR", "MONEY", "LONG VARBINARY", or "CHAR() FOR BIT DATA".
- **PRECISION** (Integer) - The precision of the column or data source. NULL is returned for data types where precision is not applicable.
- **LENGTH** (Integer) - The length in bytes of data transferred on an ODBC_GET_DATA@ or ODBC_FETCH@ if SQL_C_DEFAULT is specified. For numeric data, this size may be different from the size stored on the data source. This value is the same as **PRECISION** column for character or binary data.
- **SCALE** (Integer) - The scale of the column on the data source. NULL is returned for data types where scale is not applicable.
- **PSEUDO_COLUMN** - Indicates whether the column is a pseudo-column, such as Oracle ROWID. This contains one of the following values: SQL_PC_UNKNOWN, SQL_PC_PSEUDO, SQL_PC_NOT_PSEUDO.

  Once the application retrieves values for SQL_BEST_ROWID, the application can use these values to reselect that row within the defined scope. The **SELECT** statement is guaranteed to return either no rows or one row.

  If an application reselects a row based on the rowid column or columns and the row is not found, then the application can assume that the row was deleted or the rowid columns were modified. The opposite is not true; even if the rowid has not changed, the other columns in the row may have changed.

  Columns returned for column type SQL_BEST_ROWID are useful for applications that need to scroll forwards and backwards within a result set to retrieve the most recent data from a set of rows. The column or columns of the rowid are guaranteed not to change while positioned on that row.

  The column or columns of the rowid may remain valid even when the cursor is not positioned on the row; the application can determine this by checking the SCOPE column in the result set.

  Columns returned for column type SQL_ROWVER are useful for applications that need the ability to check if any columns in a given row have been updated while the row was

reselected using the rowid. For example, after reselecting a row using rowid, the application can compare the previous values in the SQL_ROWVER columns to the ones just fetched. If the value in a SQL_ROWVER column differs form the previous value, the application can alert the user that data on the display has changed.

See also ODBC_FETCH@, ODBC_GET_DATA@, ODBC_COLUMNS@

ODBC_STATISTICS@

Retrieves information about a table

Format ODBC_STATISTICS@(format sql_channel@ channel, stmtHandle, tableQualifier, tableOwner, tableName, tableType, Unique, Accuracy)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

tableQualifier — Qualifier name. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, and empty empty string ("") denotes those tables that do not have qualifiers.

tableOwner — String search pattern for owner names.

tableName — String search pattern for table names. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, and empty empty string ("") denotes those tables that do not have owners

tableType — List of table types to match.

Unique — Type of index: SQL_INDEX_UNIQUE or SQL_INDEX_ALL.

Accuracy — The importance of the CARDINALITY and PAGES columns in the result set:

SQL_ENSURE requests that the driver unconditionally retrieve the statistics.

SQL_QUICK requests that the driver retrieve results only if they are readily available from the server. In this case, the driver does not ensure that the values are current.

Description ODBC_STATISTICS@ returns information about a single table as a standard result set, ordered by NON-UNIQUE, TYPE, INDEX_QUALIFIER, INDEX_NAME, and SEQ_IN_INDEX. The result set combines statistics information about each index. The following lists the columns in the result set.

TABLE_QUALIFIER is the Table qualifier identifier. This is NULL if not applicable to the data source. If a driver supports qualifiers for some tables and not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string "" for those tables that do not have qualifiers.

TABLE_OWNER is the Table owner identifier. This is NULL if not applicable to the data source. If a driver supports qualifiers for some tables and not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have owners.

TABLE_NAME is the Table identifier.

NON-UNIQUE indicates whether the index prohibits duplicate values. TRUE if index values can be duplicated. FALSE if index values must be unique. NULL if TYPE is SQL_TABLE_STAT.

INDEX_QUALIFIER is the identifier that is used to qualify the index name doing a DROP INDEX. NULL is returned if the index is not supported by the data source, or if TYPE is SQL_TABLE_STAT. If a non-null value is returned in this column, it must be used to qualify the index name on a **DROP_INDEX** statement; otherwise the TABLE_OWNER name should be used to qualify the index name.

INDEX_NAME is the index identifier. NULL if TYPE is SQL_TABLE_STAT.

TYPE is the type of information being returned. This should be one of the following values:

- SQL_TABLE_STAT indicates a statistic for the table.
- SQL_TABLE_CLUSTERED indicates a clustered index.
- SQL_TABLE_HASHED indicates a hashed index.
- SQL_TABLE_OTHER indicates another type of index.

SEQ_IN_INDEX is the column sequence number in the index (starting with 1.) NULL is returned if TYPE is SQL_TABLE_STAT.

COLUMN_NAME is the column identifier. If the column is based on an expression, such as SALARY + BENEFITS, the expression is returned. If the expression cannot be determined, an empty string is returned. If the index is a filtered index, each column in the filter condition is returned. This may require more than one row. NULL is returned if TYPE is SQL_TABLE_STAT.

COLLATION is the sort sequence for the column: "A" for ascending, "D" for descending. NULL is returned if the column sort sequence is not supported by the data source, or if TYPE is SQL_TABLE_STAT.

CARDINALITY is the cardinality of the table or index. This contains the number of rows in the table if TYPE is SQL_TABLE_STAT. It contains the number of unique values in the index if TYPE is not SQL_TABLE_STAT. NULL is returned if the column sort sequence is not supported by the data source.

PAGES is the number of pages used to store the index or table. It contains the number of pages for the table if TYPE is SQL_TABLE_STAT. It contains the number of pages for the index if TYPE is not SQL_TABLE_STAT. NULL is returned if the value is not applicable to the data source or not available from the data source.

FILTER_CONDITIONS is a filter condition for the index, such as SALARY > 30000. If the filter condition cannot be determined, this is an empty string. This is NULL if the index is not a filtered index, or if it cannot be determined whether the index is a filtered index, or TYPE is SQL_TABLE_STAT.

If the row in the result set corresponds to a table, the driver sets the TYPE to SQL_TABLE_STAT and sets NON_UNIQUE, INDEX_QUALIFIER, INDEX_NAME, SEQ_IN_INDEX, COLUMN_NAME, and COLLATION to NULL. If CARDINALITY or PAGES are not available from the data source, the driver sets them to NULL.

See also ODBC_BIND_COL@, ODBC_TABLES@

ODBC_START_SERVER@

Starts the Applixware ODBC Gateway

Format ODBC_START_SERVER@(hostName)

Arguments

hostName	a string containing the name of the machine containing the elfodbc executable.

Description Executes the elfodbc executable. If hostName is an empty string (""), the elfodbc executable on the local machine is run. This executable must be in the Applixware system directory. This directory is returned by the macro SYSTEM_DIR@.

Internally, the executable is run using the macro axnet_start_slaved_process@. The Applixware Task ID of the slaved process is returned by the ODBC_START_SERVER@.

ODBC_START_SERVER@ must be run before any other ODBC macro can be successfully run. This macro must be immediately followed by the macro ODBC_ALLOC_ENV@.

See also ODBC_ALLOC_ENV@, ODBC_ALLOC_CONNECT@

ODBC_TABLES@

Establishes a result set of table names from a data source

Format ODBC_TABLES@(format sql_channel@ channel, stmtHandle, tableQualifier, tableOwner, tableName, tableType)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

tableQualifier — Qualifier name. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, and empty empty string ("") denotes those tables that do not have qualifiers.

tableOwner — String search pattern for owner names.

tableName — String search pattern for table names. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, and empty empty string ("") denotes those tables that do not have owners

tableType — List of table types to match.

Description Establishes a result set of table names that are stored in a specific database. The result set contains the following columns:

TABLE_QUALIFIER is the Table qualifier identifier. This is NULL if not applicable to the data source. If a driver supports qualifiers for some tables and not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string "" for those tables that do not have qualifiers.

TABLE_OWNER is the Table owner identifier. This is NULL if not applicable to the data source. If a driver supports qualifiers for some tables and not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have owners.

TABLE_NAME is the Table identifier.

TABLE_TYPE is one of the following:

TABLE	VIEW	SYSTEM TABLE
ALIAS	SYNONYM	GLOBAL TEMPORARY

LOCAL TEMPORARY

Data-source specific type identifiers could also be returned.

REMARKS contains a description of the table.

Once the result set is established, an application can call ODBC_BIND_COL@ to establish columns to return, and ODBC_FETCH@ to return those columns.

ODBC_TABLE_PRIVILEGES@

Establishes a result set of table privileges

Format ODBC_TABLES@(format sql_channel@ channel, stmtHandle, tableQualifier, tableOwner, tableName, tableType)

Arguments

channel — an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

- host — A string containing the name of the ODBC gateway
- id — An Applixware task ID as returned by ODBC_START_SERVER@

stmtHandle — An SQL statement handle as returned by ODBC_ALLOC_STMT@.

tableQualifier — Qualifier name. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, and empty empty string ("") denotes those tables that do not have qualifiers.

tableOwner — String search pattern for owner names.

tableName — String search pattern for table names. If a driver supports qualifiers for some tables but not for others, such as when a driver retrieves data from different DBMSs, and empty empty string ("") denotes those tables that do not have owners

tableType — List of table types to match.

Description Establishes a result set of tables and the privileges associated with each table. The result set is ordered by TABLE_QUALIFIER, TABLE_OWNER, TABLE_NAME, and PRIVILEGE. The result set contains the following columns:

TABLE_QUALIFIER is the table qualifier identifier. This is NULL if not applicable to the data source. If a driver supports qualifiers for some tables and not for others, such as when the driver retrieves data from different DBMSs, it returns an empty string "" for those tables that do not have qualifiers.

TABLE_OWNER is the Table owner identifier. This is NULL if not applicable to the data source. If a driver supports qualifiers for some tables and not for others, such as when

the driver retrieves data from different DBMSs, it returns an empty string ("") for those tables that do not have owners.

TABLE_NAME is the Table identifier.

GRANTOR is the identifier of the user who granted the privilege. This is NULL if not applicable to the data source.

GRANTEE is the identifier of the user to whom the privilege was granted.

PRIVILEGE identifies the column privilege. This may be one of the following values:

- SELECT means the grantee is permitted to retrieve data for the column
- INSERT means the grantee is permitted to provide data for the column in new rows that are inserted into the associated table.
- UPDATE means the grantee is permitted to update data in the column.
- DELETE means the grantee is permitted to delete rows from the table.
- REFERENCE means the grantee is permitted to refer to the column within a constraint (for example, a unique, referential, or table check constraint).

The scope of action permitted the grantee by a given table privilege is data source-dependent. For example, the UPDATE privilege might permit the grantee to update all columns in a table on one data source and only those columns for which the grantor has the UPDATE privilege on another data source.

IS_GRANTABLE indicates whether the grantee is permitted to grant the privileges to other users; "YES", "NO", or NULL if unknown or not applicable to the data source.

Once the result set is established, an application can call ODBC_BIND_COL@ to establish columns to return, and ODBC_FETCH@ to return those columns.

See also ODBC_BIND_COL@, ODBC_FETCH@, ODBC_TABLES@

ODBC_TRANSACT@

Requests a commit or rollback of all transactions on a statement handle

Format ODBC_TRANSACT@(format sql_channel@ channel, dbHandle, Request)

Arguments channel an array of format sql_channel@. The sql_channel@ format is defined in dbase_.am. It contains the following information:

host A string containing the name of the ODBC gateway

	id	An Applixware task ID as returned by ODBC_START_SERVER@
dbHandle	A connection handle. This is an element in the array returned by ODBC_ALLOC_CONNECT@. Use this equation to extract the handle: retval = ODBC_ALLOC_CONNECT@(channel) dbHandle = retval[1, 0]	
Option	One of the following values: SQL_COMMIT, SQL_ROLLBACK	

Description If dbHandle is SQL_NULL_HDBC, the Driver manager attempts to commit or rollback transactions on all database handles that are in a connected state. The Driver Manager calls ODBC_TRANSACT@ in the driver associated with each database handle. The Driver Manager returns SQL_SUCCESS only if it receives SQL_SUCCESS for outstanding database handle. If the Driver Manager receives SQL_ERROR on one or more database handles, it returns SQL_ERROR to the application. To determine which connection failed, during the operation, call ODBC_ERROR@ for each dbHandle.

If dbHandle is a valid connection handle, the Driver Manager calls ODBC_TRANSACT@ in the driver for the given database handle.

If Request is SQL_COMMIT, ODBC_TRANSACT@ issues a commit request for all active operations on any statement handle associated with an affected dbHandle. If Request is SQL_ROLLBACK, ODBC_TRANSACT@ issues a rollback request for all active operations on any statement handle associated with an affected dbHandle. If no transactions are active, ODBC_TRANSACT@ returns SQL_SUCCESS with no effect on any data sources.

If the driver is in manual-commit mode, (by calling ODBC_SET_CONNECT_OPTION with the ODBC_AUTOCOMMIT option set to zero), a new transaction is implicitly started when an SQL statement that can be contained with a transactionis executed against the current data source.

To determine how transaction operation affect cursors, an application calls ODBC_GET_INFO@ with the SQL_CURSOR_ROLLBACK_BEHAVIOR and SQL_CURSOR_COMMIT_BEHAVIOR options.

If the SQL_CURSOR_ROLLBACK_BEHAVIOR or SQL_CURSOR_COMMIT_BEHAVIOR value equals SQL_CB_DELETE, ODBC_TRANSACT@ closes and deletes all open cursors on all statement handles associated with the database handle and discards all pending results.

If the SQL_CURSOR_ROLLBACK_BEHAVIOR or SQL_CURSOR_COMMIT_BEHAVIOR value equals SQL_CB_CLOSE, ODBC_TRANSACT@ closes all open cursors on all statement associated with the database handle. ODBC_TRANSACT@ leaves any statement handle present in a prepared state; the application can call ODBC_EXECUTE@ for a statement handle associated with a database handle without first calling ODBC_PREPARE@.

If the SQL_CURSOR_ROLLBACK_BEHAVIOR or SQL_CURSOR_COMMIT_BEHAVIOR value equals SQL_CB_PRESERVE. ODBC_TRANSACT@ does not affect open cursors associated with a database handle. Cursors remain at the row that they pointed to prior to the call to ODBC_TRANSACT@.

For drivers and data sources that support transactions, calling ODBC_TRANSACT with either SQL_COMMIT or SQL_ROLLBACK when no transaction is active will return SQL_SUCCESS, indicating that there is no work to be committed or rolled back, and have no effect on the data source.

Drivers and data sources that do not support transactions (ODBC_GET_INFO Option SQL_TXN_CAPABLE is 0) are effectively always in autocommit more. Therefore, calling ODBC_TRANSACT@ with SQL_COMMIT will return SQL_SUCCESS. However, calling ODBC_TRANSACT@ with SQL_ROLLBACK will result in SQLSTATE S1C00 (Driver not capable), indicating that a rollback can never be performed.

Description ODBC_GET_INFO@, ODBC_FREE_STMT@

ODBC CATALOG FUNCTIONS

Description The following functions are referred to as catalog functions. These ODBC functions returns information about a data source's *catalog*. A catalog is a portion of a database containing information that describes the structure of the data in that database.

ODBC_TABLES@
ODBC_COLUMNS@
ODBC_COLUMN_PRIVILEGES@
ODBC_PRIMARY_KEYS@
ODBC_STATISTICS@
ODBC_PROCEDURES@
ODBC_PROCEDURE_COLUMNS@
ODBC_TABLE_PRIVILEGES@
ODBC_FOREIGN_KEYS@
ODBC_SPECIAL_COLUMNS@

SQL_COLUMN_AUTO_INCREMENT

Description TRUE if the column is autoincrement. FALSE if the column is not autoincrement or is not numeric.

Auto increment is valid for numeric data type columns only. An application can insert values into an autoincrement column, but cannot update values in the column.

See also ODBC_COL_ATTRIBUTES@

SQL_COLUMN_CASE_SENSITIVE

Description TRUE if the column is treated as case sensitive. FALSE if the column is not case sensitive for collations and comparisons, or is non-character.

See also ODBC_COL_ATTRIBUTES@

SQL_COLUMN_COUNT

Description Number of columns available in the result set. The *icol* argument is ignored.

See also ODBC_COL_ATTRIBUTES@

SQL_COLUMN_DISPLAY_SIZE

Description Maximum number of characters required to display data from the column.

See also ODBC_COL_ATTRIBUTES@

SQL_COLUMN_LABEL

Description The column label or title. For example, a column named EMPName might be labeled Employee Name.

If a column does not have a label, the column name is returned. If the column is unlabeled and unnamed, an empty string is returned.

See also ODBC_COL_ATTRIBUTES@

SQL_COLUMN_LENGTH

Description The length in bytes of data transferred on an ODBC_GET_DATA@ or an ODBC_FETCH@. operation if SQL_C_DEFAULT is specified. For numberic data this size may be different than the size of the data stored on the data source.

See also ODBC_COL_ATTRIBUTES@

SQL_COLUMN_MONEY

Description TRUE if the column is money data type. FALSE if the column is not money data type.

See also ODBC_COL_ATTRIBUTES@

SQL_COLUMN_NAME

Description The column name. If the column is unnamed, an empty string is returned.

See also ODBC_COL_ATTRIBUTES@

SQL_COLUMN_NULLABLE

Description SQL_NO_NULLS if the column does not accept NULL values. SQL_NULLABLE if the column accepts NULL values. SQL_NULLABLE_UNKNOWN if it is not knows if the column accepts NULL values.

SQL_COLUMN_OWNER_NAME

Description The owner of the table that contains the column. The returned value is implementation-defined if the column is an expression or if the column is part of a view. If the data source does not support owners, or the owner cannot be determined, an empty string is returned.

See also ODBC_COL_ATTRIBUTES@

SQL_COLUMN_PRECISION

Description The precision of the column on the data source.

See also ODBC_COL_ATTRIBUTES@

SQL_COLUMN_QUALIFIER_NAME

Description The qualifier of the table that contains the column. The returned value is implementation-defined if the column is an expression or if the column is part of a view. If the data source does not support qualifiers, or the qualifier cannot be determined, an empty string is returned.

See also ODBC_COL_ATTRIBUTES@

SQL_COLUMN_SCALE

Description The scale of the column on the data source.

See also ODBC_COL_ATTRIBUTES@

SQL_COLUMN_SEARCHABLE

Description SQL_UNSEARCHABLE if the column cannot be used in a **WHERE** clause.

SQL_LIKE_ONLY if the column can be used in a **WHERE** clause only with the **LIKE** predicate.

SQL_ALL_EXCEPT_LIKE if the column can be used in a **WHERE** clause with all comparison operators except **LIKE**.

SQL_SEARCHABLE if the column can be used in a **WHERE** clause with any comparison operators.

See also ODBC_COL_ATTRIBUTES@

SQL_COLUMN_TABLE_NAME

Description The name of the table that contains the column. The returned value is implementation-defined if the column is an expression or if the column is part of a view.

See also ODBC_COL_ATTRIBUTES@

SQL_COLUMN_TYPE

Description SQL data type. This can be ODBC SQL data type or a driver-specific SQL data type.

See also ODBC_COL_ATTRIBUTES@

SQL_COLUMN_TYPE_NAME

Description Data-source-dependent data type name. For example, "CHAR", "VARCHAR", "MONEY", "LONG VARBINARY", or "CHAR() FOR BIT DATA".

See also ODBC_COL_ATTRIBUTES@

SQL_COLUMN_UNSIGNED

Description TRUE if the column is unsigned (or not numeric). FALSE if the column is signed.

See also ODBC_COL_ATTRIBUTES@

SQL_COLUMN_UPDATEABLE

Description Column is defined by the values for the defined constants:

SQL_ATTR_READONLY

SQL_ATTR_WRITE

SQL_ATTR_READWRITE_UNKNOWN

SQL_COLUMN_UPDATEABLE describes the updateability of the column in the result set. Whether a column is updateable can be based on the data type, user privileges, and the definition of the data set itself. If it is unclear whether a column is updateable, SQL_ATTR_READWRITE_UNKNOWN should be returned.

See also ODBC_COL_ATTRIBUTES@

SQL_ACCESSIBLE_PROCEDURES

Description A character string. "Y" if the user can execute all procedure returned by ODBC_PROCEDURES@, "N" if there may be procedures returned that the user cannot execute.

See also ODBC_GET_INFO@

SQL_ACCESSIBLE_TABLES

Description A character string. Y means user is guaranteed **SELECT** privileges to all tables returned by ODBC_TABLES@. N means there may be tables returned that the user cannot access.

See also ODBC_GET_INFO@

SQL_ACTIVE_CONNECTIONS

Description A number specifiying the maximum number of database connection handles that the driver can support. This value can reflect a limitation imposed by the driver or the data source. If there is no specified limit, or the limit is unknown, this value is set to zero.

See also ODBC_GET_INFO@

SQL_ACTIVE_STATEMENTS

Description The number of active statement handles that the driver can support on a database handle. This value can reflect a limitation imposed by the driver or the data source. If there is no specified limit, or the limit is unknown, this value is set to zero.

See also ODBC_GET_INFO@

SQL_ALTER_TABLE

Description A 32-bit bitmask enumerating the clauses in the alter table statement supported by the data source.

The following bitmasks are used to determine which clauses are supported:

- SQL_AT_ADD_COLUMN
- SQL_AT_DROP_COLUMN

See also ODBC_GET_INFO@

SQL_BOOKMARK_PERSISTANCE

Description A 32-bit bitmask enumerating the operations through which bookmarks persist. The following bitmasks are used in conjunction with the flag returned by ODBC_GET_INFO@ to determine the operations through which bookmarks persist:

SQL_BP_CLOSE means bookmarks are valid after an application calls ODBC_FREE_STMT@ with the SQL_CLOSE option to close the cursor on a statement handle.

SQL_BP_DELETE means the bookmark for a row is valid after that row has been deleted.

SQL_BP_DROP means bookmarks are valid after an application calls ODBC_FREE_STMT@ with the SQL_DROP option to drop a statement handle.

SQL_BP_SCROLL means bookmarks are valid after any scrolling operation (ODBC_EXTENDED_FETCH@). Because all bookmarks must remain valid after ODBC_EXTENDED_FETCH@ is called, this value can be used by applications to determine if bookmarks are supported.

SQL_BP_TRANSACTION means bookmarks are valid after an application commits or rolls back a transaction.

SQL_BP_UPDATE means the bookmark for row is valid after any column in that row has been updated, including key columns.

SQL_BP_OTHER_HSTMT means a bookmark associated with one statement handle can be used with another statement handle.

See also ODBC_GET_INFO@

SQL_COLUMN_ALIAS

Description A character string. Y means the data source supports column aliases. N means the data source does not support column aliases.

See also ODBC_GET_INFO@

SQL_CONCAT_NULL_BEHAVIOR

Description An integer value indicating how the data source handles the concatenation of NULL valued character data type columns with non-NULL valued character data type columns.

SQL_CB_NULL means the result is NULL valued.

SQL_CB_NON_NULL means the result is concatenation of non-NULL valued column or columns.

See also ODBC_GET_INFO@

SQL_CONVERT_*

Description A 32-bit bitmask. The bitmask indicates the conversions supported by the data source with the convert scalar function for data of the type named in the infoType argument. If the bitmask equals zero, the data source does not support any conversions for data of the named type, including conversion to the same data type.

For example, to find out if a data source supports the conversion of SQL_INTEGER data to the SQL_BIGINT data type, an application calls ODBC_GET_INFO@ with the infoType of SQL_CONVERT_INTEGER. The application ANDs the returned bitmask with SQL_CVT_BIGINT. If the resulting value is non-zero, the conversion is supported.

The following bitmasks are used to determine which conversions are supported.

SQL_CVT_BIGINT SQL_CVT_BINARY SQL_CVT_BIT
SQL_CVT_CHAR SQL_CVT_DATE SQL_CVT_DECIMAL
SQL_CVT_DOUBLE SQL_CVT_FLOAT SQL_CVT_INTEGER
SQL_CVT_LONGVARBINARY SQL_CVT_LONGVARCHAR
SQL_CVT_NUMERIC SQL_CVT_REAL
SQL_CVT_SMALLINT SQL_CVT_TIME
SQL_CVT_TIMESTAMP SQL_CVT_TINYINT
SQL_CVT_VARBINARY SQL_CVT_VARCHAR

See also ODBC_GET_INFO@

SQL_CONVERT_FUNCTIONS

Description A 32-bit bitmask enumerating the scalar conversion functions supported by the driver and the associated data source.

See also ODBC_GET_INFO@

SQL_CORRELATION_NAME

Description A 16-bit integer indicating if table correlation names are supported.

SQL_CN_NONE means correlation names are not supported.

SQL_CN_DIFFERENT means correlation names are supported, but must differ from the names of the tables they represent.

SQL_CN_ANY means correlation names are supported, and can be any valid user-defined name.

See also ODBC_GET_INFO@

SQL_CURSOR_COMMIT_BEHAVIOR

Description A 16-bit integer value indicating how a **COMMIT** operation affect cursors and prepared statements in the data source.

SQL_CB_DELETE means close cursors and delete prepared statements. To use the cursor again, the application must re-prepare and re-execute the statement handle.

SQL_CB_CLOSE means close cursors. For prepared statements, the application can call ODBC_EXECUTE@ on the statement handle without calling ODBC_PREPARE@ again.

SQL_CB_PRESERVE means preserve cursors in the same position as before the **COMMIT** operation. The application can continue to fetch data, or it can close the cursor and re-execute the statement handle without re-preparing it.

See also ODBC_GET_INFO@

SQL_CURSOR_ROLLBACK_BEHAVIOR

Description A 16-bit integer value indicating how a **ROLLBACK** operation affect cursors and prepared statements in the data source.

SQL_CB_DELETE means close cursors and delete prepared statements. To use the cursor again, the application must re-prepare and re-execute the statement handle.

SQL_CB_CLOSE means close cursors. For prepared statements, the application can call ODBC_EXECUTE@ on the statement handle without calling ODBC_PREPARE@ again.

SQL_CB_PRESERVE means preserve cursors in the same position as before the **ROLLBACK** operation. The application can continue to fetch data, or it can close the cursor and re-execute the statement handle without re-preparing it.

See also ODBC_GET_INFO@

SQL_DATA_SOURCE_NAME

Description The character string with the data source name used during connection. If the application called ODBC_CONNECT@, this is the value of the source argument.

If the application called ODBC_DRIVER_CONNECT@ or ODBC_BROWSE_CONNECT@, this is the value of the DSN keyword in the connString argument. If the connection string did not contain a DSN keyword (such as when it contains the DRIVER keyword), this is an empty string.

See also ODBC_GET_INFO@

SQL_DATA_SOURCE_READ_ONLY

Description A character string. Y means the data source is set to Read Only. N means the data source is Read/Write.

This characteristic pertains only to the data source itself. It is not a characteristic of the driver that enables access to the data source.

See also ODBC_GET_INFO@

SQL_DATABASE_NAME

Description A character string with the name of the current database in use, if the data source defines a named object called "database."

This information type is not used in ODBC 2.0. It has been replaced by SQL_CURRENT_QUALIFIER.

See also ODBC_GET_INFO@

SQL_DBMS_NAME

Description A character string with the name of the DBMS product accessed by the driver.

See also ODBC_GET_INFO@

SQL_DBMS_VER

Description A character string indicating the version of the DBMS product accessed by the driver. The version is of the form ##.##.####, where the first two digits are the major revision, the next two digits are the minor revision, and the last four digits are the release version. The driver must render the DBMS product-specific version as well. For example, "04.02.0000 Rdb 4.1".

See also ODBC_GET_INFO@

SQL_DEFAULT_TXN_ISOLATION

Description Transaction isolation is explained in detail in the discussion of the macro ODBC_SET_CONNECT_OPTION@.

SQL_DRIVER_HDBC

Description A 32-bit value, the driver's environment handle or connection handle, determined by the dbHandle argument.

This information type is implemented by the driver manager alone.

See also ODBC_GET_INFO@

SQL_DRIVER_HENV

Description A 32-bit value, the driver's environment handle or connection handle, determined by the dbHandle argument.

This information type is implemented by the driver manager alone.

See also ODBC_GET_INFO@

SQL_DRIVER_HLIB

Description A 32-bit value, the library handle returned to the Driver Manager when it loaded the driver shared library. The handle is only valid for the dbHandle specified in the call to ODBC_GET_INFO@.

See also ODBC_GET_INFO@

SQL_DRIVER_HSTMT

Description A 32-bit value. This is the driver's statement handle determined by the Driver Manager statement handle, which must be passed on input in rgbInfoValue from the application. Note that in this case, *rgbInfoValue* is both an input and an output parameter argument. The input statement handle passed in *rgbInfoValue* must have a statement handle established for the database handle.

This information type is implemented by the Driver Manager alone.

See also ODBC_GET_INFO@

SQL_DRIVER_NAME

Description A character string with the filename of the driver used to access the data source.

See also ODBC_GET_INFO@

SQL_DRIVER_ODBC_VER

Description A character string with the version of ODBC that the driver supports. The version is of the form ##.##, where the first two digits are the major version and the next two digits are the minor version. SQL_SPEC_MAJOR and SQL_SPEC_MINOR define the major and minor version numbers. The driver should return "02.00".

See also ODBC_GET_INFO@

SQL_DRIVER_VER

Description A character string with the version of the driver and, optionally, a description of the driver. At a minimum, the version is of the form ##.##.####, where the first two digits are the major revision, the next two digits are the minor revision, and the last four digits are the release version.

See also ODBC_GET_INFO@

SQL_EXPRESSION_IN_ORDERBY

Description A character string. Y if the data source supports expressions in the **ORDER BY** list; N if it does not.

See also ODBC_GET_INFO@

SQL_FETCH_DIRECTION

Description A 32-bit bitmask enumerating the supported fetch direction options. The following bitmasks are used in conjunction with the flag to determine which options are supported:

SQL_FD_FETCH_NEXT	SQL_FD_FETCH_FIRST
SQL_FD_FETCH_LAST	SQL_FD_FETCH_PRIOR
SQL_FD_FETCH_ABSOLUTE	SQL_FD_FETCH_RELATIVE
SQL_FD_FETCH_RESUME	SQL_FD_FETCH_BOOKMARK

See also ODBC_GET_INFO@

SQL_FILE_USAGE

Description A 16-bit integer value indicating how a single-tier driver directly treats files in a data source. This is one of the following values:

SQL_FILE_NOT_SUPPORTED - the driver is not a single-tier driver.

SQL_FILE_TABLE - A single-tier driver treats files in a data source as tables.

SQL_FILE_QUALIFIER - A single-tier driver treats files in a data source as a qualifier.

See also ODBC_GET_INFO@

SQL_GETDATA_EXTENSIONS

Description A 32-bit bitmask enumerating extensions to ODBC_GET_DATA@.

The following bitmasks are used in conjunction with the flag to determine what common extensions the driver supports for ODBC_GET_DATA@:

SQL_GD_ANY_COLUMN means ODBC_GET_DATA@ can be called for any unbound column, including those before the last bound column. Note that the columns must be called in order of ascending column number unless SQL_GD_ANY_ORDER is also returned.

SQL_GD_ANY_ORDER means ODBC_GET_DATA@ can be called for unbound columns in any order. Note that ODBC_GET_DATA@ can only be called for columns after the last bound column unless SQL_GD_ANY_COLUMN is also returned.

SQL_GD_BLOCK means ODBC_GET_DATA@ can be called for an unbound column in any row in a block (more than one row) of data after positioning to that row of data with ODBC_SET_POS@.

SQL_GD_BOUND means ODBC_GET_DATA@ can be called for bound columns as well as unbound columns. A driver cannot return this value unless it also returns SQL_GD_ANY_COLUMN.

ODBC_GET_DATA@ is is only required to return data from unbound columns that occur after the last bound column. These are called in order of increasing column number , and are not in a row within a block of rows.

See also ODBC_GET_INFO@

SQL_GROUP_BY

Description An integer value specifying the relationship between the columns in the **GROUP BY** clause and the non-aggregated columns in the select list:

SQL_GB_NOT_SUPPORTED means **GROUP BY** clauses are not supported.

SQL_GB_GROUP_BY_EQUALS_SELECT means the **GROUP BY** clause must contain all non-aggregated columns in the select list. It cannot contain any other columns. For example, **SELECT DEPT, MAX(SALARY) FROM EMPLOYEE GROUP BY DEPT**.

SQL_GB_GROUP_BY_CONTAINS_SELECT means the **GROUP BY** clause must contain all non-aggregated columns in the select list. It can contain columns that are not int he select list. For example, **SELECT DEPT, MAX(SALARY) FROM EMPLOYEE GROUP BY DEPT, AGE.**

SQL_GB_NO_RELATION means the columns in the **GROUP BY** clause and the select list are not related. The meaning of non-grouped, non-aggregated columns in the select list is data-source dependent. For example, **SELECT DEPT, SALARY FROM EMPLOYEE GROUP BY DEPT, AGE.**

See also ODBC_GET_INFO@

SQL_IDENTIFIER_CASE

Description An integer value as follows:

SQL_IC_UPPER means identifiers in SQL are case insensitive and are stored in upper case in system catalog.

SQL_IC_LOWER means identifiers in SQL are case insensitive and are stored in lower case in system catalog.

SQL_IC_SENSITIVE means identifiers in SQL are case sensitive and are stored in mixed case in system catalog.

SQL_IC_MIXED means identifiers in SQL are case insensitive and are stored in mixed case in system catalog.

See also ODBC_GET_INFO@

SQL_IDENTIFIER_QUOTE_CHAR

Description The character string used as the starting and ending delimiter of a quoted (delimited) identifier in SQL statements. (Identifiers passed as arguments to ODBC macros do not need to be quoted.) If the data source does not support quoted identifiers, a blank is returned.

See also ODBC_GET_INFO@

SQL_KEYWORDS

Description A character string containing a comma-separated list of all data source-specific keywords. This list does not contain keywords specific to ODBC or keywords used by both the data source and ODBC.

See also ODBC_GET_INFO@

SQL_LIKE_ESCAPE_CLAUSE

Description A character string: Y if the data source supports an escape character for the percent (%) character and underscore character (_) in a **LIKE** predicate and the driver supports the ODBC syntax for defining a **LIKE** predicate escape character; N otherwise.

See also ODBC_GET_INFO@

SQL_LOCK_TYPES

Description A bitmask enumerating the supported lock types for the fLock argument in ODBC_SET_POS@. The following bitmasks are used in conjunction with the flag to determine which lock types are supported:

SQL_LCK_NO_CHANGE

SQL_LCK_EXCLUSIVE

SQL_LCK_UNLOCK

See also ODBC_GET_INFO@

SQL_MAX_BINARY_LITERAL_LEN

Description An integer value specifying the maximum length of a binary literal in an SQL statement. The maximum length is the number of hexadecimal characters , excluding the literal prefix and suffix returned by ODBC_GET_TYPE_INFO@. For example, the binary literal 0xFFAA has a length of 4. If there is no maximum length or the length is unknown, this value is set to zero.

See also ODBC_GET_INFO@

SQL_MAX_CHAR_LITERAL_LEN

Description An integer value specifying the maximum length of a character literal in an SQL statement. The maximum length is the number of characters , excluding the literal prefix and

suffix returned by ODBC_GET_TYPE_INFO@. If there is no maximum length or the length is unknown, this value is set to zero.

See also ODBC_GET_INFO@

SQL_MAX_COLUMN_NAME_LEN

Description An integer value specifying the maximum length of a column name in the data source. If there is no maximum length or the length is unknown, this value is set to zero.

See also ODBC_GET_INFO@

SQL_MAX_COLUMNS_IN_GROUP_BY

Description An integer value specifying the maximum number of columns allowed in a **GROUP BY** clause. If there is no specified limit or the limit is not known, this value is set to zero.

See also ODBC_GET_INFO@

SQL_MAX_COLUMNS_IN_INDEX

Description An integer value specifying the maximum number of columns allowed in an index. If there is no specified limit or the limit is not known, this value is set to zero.

See also ODBC_GET_INFO@

SQL_MAX_COLUMNS_IN_ORDER_BY

Description An integer value specifying the maximum number of columns allowed in a **ORDER BY** clause. If there is no specified limit or the limit is not known, this value is set to zero.

See also ODBC_GET_INFO@

SQL_MAX_COLUMNS_IN_SELECT

Description An integer value specifying the maximum number of columns allowed in a select. If there is no specified limit or the limit is not known, this value is set to zero.

See also ODBC_GET_INFO@

SQL_MAX_COLUMNS_IN_TABLE

Description An integer value specifying the maximum number of columns allowed in a table. If there is no specified limit or the limit is not known, this value is set to zero.

See also ODBC_GET_INFO@

SQL_MAX_CURSOR_NAME_LEN

Description An integer value specifying the maximum length of a cursor name in a data source. If there is no specified limit or the limit is not known, this value is set to zero.

See also ODBC_GET_INFO@

SQL_MAX_INDEX_SIZE

Description An integer value specifying the maximum number of bytes allowed in the combined fields of an index. If there is no specified limit or the limit is not known, this value is set to zero.

See also ODBC_GET_INFO@

SQL_MAX_OWNER_NAME_LEN

Description An integer value specifying the maximum length of an owner name in a data source. If there is no specified limit or the limit is not known, this value is set to zero.

See also ODBC_GET_INFO@

SQL_MAX_PROCEDURE_NAME_LEN

Description An integer value specifying the maximum length of an procedure name in a data source. If there is no specified limit or the limit is not known, this value is set to zero.

See also ODBC_GET_INFO@

SQL_MAX_QUALIFIER_NAME_LEN

Description An integer value specifying the maximum length of an qualifier name in a data source. If there is no specified limit or the limit is not known, this value is set to zero.

See also ODBC_GET_INFO@

SQL_MAX_ROW_SIZE

Description An integer value specifying the maximum length of a singkle row in a table. If there is no specified limit or the limit is not known, this value is set to zero.

See also ODBC_GET_INFO@

SQL_MAX_ROW_SIZE_INCLUDES_LONG

Description A character string: Y if the maximum row size returned for the SQL_MAX_ROW_SIZE information type includes the length of all SQL_LONGVARCHAR and SQL_LONGVARBINARY columns in the row; N otherwise.

See also ODBC_GET_INFO@

SQL_MAX_STATEMENT_LEN

Description An integer value specifying the maximum length (number of characters including white space) an an SQL statement. If there is no maximum length, or the length is not known, this value is set to zero.

See also ODBC_GET_INFO@

SQL_MAX_TABLE_NAME_LEN

Description An integer value specifying the maximum length of a table name in a data source. If there is no maximum length, or the length is not known, this value is set to zero.

See also ODBC_GET_INFO@

SQL_MAX_TABLES_IN_SELECT

Description An integer value specifying the maximum number of tables allowed in the FROM clause of a SELECT statement. If there is no specified limit or the limit is unknown, this value is set to zero.

See also ODBC_GET_INFO@

SQL_MAX_USER_NAME_LEN

Description An integer value specifying the maximum length of a user name in a data source. If there is no maximum length, or the length is not known, this value is set to zero.

See also ODBC_GET_INFO@

SQL_MULT_RESULT_SETS

Description A character string: Y if the data source supports multiple result sets, N otherwise.

See also ODBC_GET_INFO@

SQL_MULTIPLE_ACTIVE_TXN

Description A character string: Y if active transaction on multiple connection are allowed; N otherwise.

See also ODBC_GET_INFO@

SQL_NEED_LONG_DATA_LEN

Description A character string: Y if the data source needs the length of a long data value (the data type is SQL_LONGVARBINARY, SQL_LONGVARCHAR, or a long, data source-specific data type) before that value is sent to the data source. N if the length is not required. For more information, see ODBC_BIND_PARAMETER@ and ODBC_SET_POS@.

See also ODBC_GET_INFO@

SQL_NON_NULLABLE_COLUMNS

Description An integer specifying whether the data source supports non-nullable columns:

SQL_NNC_NULL means all columns must be nullable

SQL_NNC_NON_NULL means columns may be non-nullable. The data source supports the **NOT NULL** columns constraint in **CREATE TABLE** statements.

See also ODBC_GET_INFO@

SQL_NULL_COLLATION

Description An integer value specifying where NULLs are sorted in a list:

SQL_NC_END means NULLs are sorted at the end of the list, regardless of the sort order.

SQL_NC_HIGH means NULLs are sorted at the high end of the list.

SQL_NC_LOW means NULLs are sorted at the low end of the list.

SQL_NC_START means NULLs are sorted at the start of the list, regardless of the sort order.

See also ODBC_GET_INFO@

SQL_NUMERIC_FUNCTIONS

Description A 32-bit bitmask enumerating the scalar numeric functions supported by the driver and associated with the data source. The following bitmasks are used to determine which numeric function are available:

SQL_FN_NUM_ABS	SQL_FN_NUM_ACOS
SQL_FN_NUM_ASIN	SQL_FN_NUM_ATAN
SQL_FN_NUM_ATAN2	SQL_FN_NUM_CEILING
SQL_FN_NUM_COS	SQL_FN_NUM_COT
SQL_FN_NUM_DEGREES	SQL_FN_NUM_EXP
SQL_FN_NUM_FLOOR	SQL_FN_NUM_LOG
SQL_FN_NUM_LOG10	SQL_FN_NUM_MOD
SQL_FN_NUM_PI	SQL_FN_NUM_POWER
SQL_FN_NUM_RADIANS	SQL_FN_NUM_RAND
SQL_FN_NUM_ROUND	SQL_FN_NUM_SIGN
SQL_FN_NUM_SIN	SQL_FN_NUM_SQRT
SQL_FN_NUM_TAN	SQL_FN_NUM_TRUNCATE

See also ODBC_GET_INFO@

SQL_ODBC_API_CONFORMANCE

Description An integer value indicating the level of ODBC conformance:

SQL_OAC_NONE = none.

SQL_OAC_LEVEL1 = Level 1 supported

SQL_OAC_LEVEL2 = Level 2 supported

See also ODBC_GET_INFO@

SQL_ODBC_SAG_CLI_CONFORMANCE

Description An integer value indicating compliance to the functions of the SAG specification:

SQL_OSCC_NOT_COMPLIANT means not SAG-compliant; one or more core functions are not supported

SQL_OSCC_COMPLIANT means SAG compliant

See also ODBC_GET_INFO@

SQL_ODBC_SQL_CONFORMANCE

Description An integer value indicating SQL grammar supported by the driver:

SQL_OSC_MINIMUM means minimum grammar is supported.

SQL_OSC_CORE means core grammar is supported

SQL_OSC_EXTENDED means extended grammar is supported

See also ODBC_GET_INFO@

SQL_ODBC_SQL_OPT_IEF

Description A character string: Y if the data source supports the optional Integrity Enhancement Facility; N if it does not.

See also ODBC_GET_INFO@

SQL_ODBC_VER

Description A character string with the version of ODBC to which the Driver Manager conforms. The version is of the form ##.##, where the first two digits are the major version, and the next two digits are the minor version. This is implemented only in the Driver Manager.

See also ODBC_GET_INFO@

SQL_ORDER_BY_COLUMNS_IN_SELECT

Description A character string: Y if the columns in the **ORDER BY** clause must be in the select list; otherwise N.

See also ODBC_GET_INFO@

SQL_OUTER_JOINS

Description A character string, as follows:

N means the data source does not support outer joins.

Y means the data source supports two-table outer joins, and the driver supports the ODBC outer join syntax except for nested outer joins. However, columns on the left side of the comparison operator in the ON clause must come from the left hand table in the outer join, and columns on the right side of the comparison operator must come from the right-hand table.

P means the data source partially supports nested outer joins, and the driver supports the ODBC outer join syntax. However, columns on the left side of the comparison operator in the ON clause must come from the left hand table in the outer join, and columns on the right side of the comparison operator must come from the right-hand table. Also, the right-hand table of an outer join cannot be included in an inner join.

F means the data source fully supports nested outer joins, and the driver supports the ODBC outer join syntax.

See also ODBC_GET_INFO@

SQL_OWNER_TERM

Description A character string with the data source vendor's name for an owner; for example, "owner", "Authorization ID", or "Schema".

See also ODBC_GET_INFO@

SQL_OWNER_USAGE

Description A bitmask enumerating the statements in which owners can be used:

SQL_OU_DML_STATEMENTS means owners are supported in all Data Manipulation Language Statements: **SELECT**, **INSERT**, **UPDATE**, **DELETE**, and, if supported, **SELECT FOR UPDATE** and positioned update and delete statements.

SQL_OU_PROCEDURE_INVOCATION means owners are supported in the ODBC procedure invocation statement.

SQL_OU_TABLE_DEFINITION means owners are supported in all table definition statements: **CREATE TABLE**, **CREATE VIEW**, **ALTER TABLE**, **DROP TABLE**, and **DROP VIEW**.

SQL_OU_INDEX_DEFINITION means owners are supported in all index definition statements: **CREATE INDEX** and **DROP INDEX**.

SQL_OU_PRIVILEGE_DEFINITION means owners are supported in all privilege definition statements: **GRANT** and **REVOKE**.

See also ODBC_GET_INFO@

SQL_POS_OPERATIONS

Description A 32-bit bitmask enumerating the supported operations in ODBC_SET_POS@. The following bitmasks are used in conjunction with the flag to determine which options are supported:

SQL_POS_POSITION	SQL_POS_REFRESH
SQL_POS_UPDATE	SQL_POS_DELETE
SQL_POS_ADD	

See also ODBC_GET_INFO@

SQL_POSITIONED_STATEMENTS

Description A 32-bit bitmask enumerating the supported positioned SQL statements. The following bitmasks are used to determine which statements are supported:

SQL_PS_POITIONED DELETE

SQL_PS_POSITIONED UPDATE

SQL_PS_SELECT_FOR_UPDATE

See also ODBC_GET_INFO@

SQL_PROCEDURE_TERM

Description A character string with the data source vendor's name for a procedure. For example, "database procedure", "stored procedure", "procedure".

See also ODBC_GET_INFO@

SQL_PROCEDURES

Description A character string: Y if the data source supports procedures and the driver supports the ODBC procedure invocation syntax; N otherwise.

See also ODBC_GET_INFO@

SQL_QUALIFIER_LOCATION

Description An integer value uindicating the position of the qualifier in a qualified table name:

SQL_QL_START

SQL_QL_END

For example, a Text driver returns SQL_QL_START because the directory (qualifier) name is at the start of the table name, as in /empdata/emp.dbf. An ORACLE Server driver returns SQL_QL_END, because the qualifier is at the end of the table name, as in ADMIN.EMP@EMPDATA.

See also ODBC_GET_INFO@

SQL_QUALIFIER_NAME_SEPARATOR

Description A character string: the character or characters that the data source defines as the separator between a qualifier name and the qualified name element that follows it.

See also ODBC_GET_INFO@

SQL_QUALIFIER_TERM

Description A character string with the data source vendor's name for a qualifier; for example, "database" or "directory".

See also ODBC_GET_INFO@

SQL_QUALIFIER_USAGE

Description A bitmask enumerating the statements in which qualifiers can be used:

SQL_QU_DML_STATEMENTS means qualifiers are supported in all Data Manipulation Language Statements: **SELECT**, **INSERT**, **UPDATE**, **DELETE**, and, if supported, **SELECT FOR UPDATE** and positioned update and delete statements.

SQL_QU_PROCEDURE_INVOCATION means owners are supported in the ODBC procedure invocation statement.

SQL_QU_TABLE_DEFINITION means owners are supported in all table definition statements: **CREATE TABLE**, **CREATE VIEW**, **ALTER TABLE**, **DROP TABLE**, and **DROP VIEW**.

SQL_QU_INDEX_DEFINITION means owners are supported in all index definition statements: **CREATE INDEX** and **DROP INDEX**.

SQL_QU_PRIVILEGE_DEFINITION means owners are supported in all privilege definition statements: **GRANT** and **REVOKE**.

See also ODBC_GET_INFO@

SQL_QUOTED_IDENTIFIER_CASE

Description An integer value as follows:

SQL_IC_UPPER means identifiers in SQL are case insensitive and are stored in upper case in system catalog.

SQL_IC_LOWER means identifiers in SQL are case insensitive and are stored in lower case in system catalog.

SQL_IC_SENSITIVE means identifiers in SQL are case sensitive and are stored in mixed case in system catalog.

SQL_IC_MIXED means identifiers in SQL are case insensitive and are stored in mixed case in system catalog.

See also ODBC_GET_INFO@

SQL_ROW_UPDATES

Description A character string: Y if a keyset-driven or mixed cursor maintains row versions or values for all fethed rows and therefore can detect any changes made to a row by any user since the row was last fetched. Otherwise, N.

See also ODBC_GET_INFO@

SQL_SCROLL_CONCURRENCY

Description A 32-bit bitmask enumerating the concurrency control options supported for scrollable cursors. The following bitmasks are used to determine which options are supported:

SQL_SCCO_READ_ONLY means the cursoir is read-only. No updates are allowed.

SQL_SCCO_LOCK means that the cursor uses the lowest levels of locking sufficient to insure that the row can be updated.

SQL_SCCO_OPT_ROWVER indicates that the cursor uses optimistic concurrency controls comparing row versions, such as SQLBase ROWID, or Sybase TIMESTAMP.

SQL_SCCO_OPT_VALUES indicates that the cursor uses optimistic concurrency controls comparing values.

See also ODBC_GET_INFO@

SQL_SCROLL_OPTIONS

Description A 32-bit bitmask enumerating the scroll options supported for scollable cursors. The following bitmasks are used to determine which options are supported:

SQL_SO_FORWARD_ONLY means the cursor only scrolls forward.

SQL_SO_STATIC means the data in the result set is static.

SQL_SO_KEYSET_DRIVEN means the driver saves and uses the keys for every row in the result set.

SQL_SO_DYNAMIC means the driver keeps the keys for every row in the rowset (the keyset size is the same as the rowset size).

SQL_SO_MIXED means the driver keeps the keys for every row in the keyset, and the keyset size is greater than the rowset size. The cursor is keyset-driven inside the keyset and dynamic outside the keyset.

See also ODBC_GET_INFO@

SQL_SEARCH_PATTERN_ESCAPE

Description A character string specifying what the driver supports as an escape character that permits the use of the pattern match metacharacters underscore (_) and percent (%) as valid characters in search patterns. This escape character applies only for those catalog function arguments that support search strings. If this string is empty, the driver does not support a search-pattern escape character. This infotype is limited to ODBC catalog functions.

See also ODBC_GET_INFO@

SQL_SERVER_NAME

Description A character string with the actual data source-specific server name. This is useful when a data source name is used during an ODBC_CONNECT@.

See also ODBC_GET_INFO@

SQL_SPECIAL_CHARACTERS

Description A character string containing all special characters (that is, all characters except a through z, A through Z, 0 through 9, and underscore) that can be used in an object name, such as a table, column, or index name, on the data source. For example, "#$^".

See also ODBC_GET_INFO@

SQL_STATIC_SENSITIVITY

Description A 32-bit bitmask enumerating whether changes made by an application to a static or keyset-driven cursor throgh ODBC_SET_POS@ or positioned update or delete statements can be detected by that application:

SQL_SS_ADDITIONS means added rows are visible to the cursor; the cursor can scroll to these rows. Where these rows are added to the cursor is driver-dependent.

SQL_SS_DELETIONS means deleted rows are no longer available to the cursor and do not leav a "hole" in the result set; after the cursor scrolls from a deleted row, it cannot return to that row.

SQL_SS_UPDATES means updates to rows are visible to the cursor; If the cursor scrolls from and returns to an updated row, the data returned by the cursor is the updated data, not the original data. Because updating key values in a keyset-driven cursor is considered to be deleting the existing row and adding a new row, this value is always returned for keyset-driven cursors.

Whether an application can detect changes made to the result set by other users, including other cursors in the same application, depends on the cursor type.

See also ODBC_GET_INFO@

SQL_STRING_FUNCTIONS

Description A 32-bit bitmask enumerating the scalar string functions supported by the driver and associated data source. The following bitmasks are used to determine which string functions are supported:

SQL_FN_STR_ASCII SQL_FN_STR_CHAR

SQL_FN_STR_CONCAT SQL_FN_STR_DIFFERENCE

SQL_FN_STR_INSERT SQL_FN_STR_LCASE
SQL_FN_STR_LEFT SQL_FN_STR_LENGTH
SQL_FN_STR_LOCATE SQL_FN_STR_LOCATE_2
SQL_FN_STR_LTRIM SQL_FN_STR_REPEAT
SQL_FN_STR_REPLACE SQL_FN_STR_RIGHT
SQL_FN_STR_RTRIM SQL_FN_STR_SOUNDEX
SQL_FN_STR_SPACE SQL_FN_STR_SUBSTRING
SQL_FN_STR_UCASE

See also ODBC_GET_INFO@

SQL_SUBQUERIES

Description A 32-bit bitmask enumerating the predicates that support subqueries:

SQL_SQ_CORRELATED_SUBQUERIES
SQL_SQ_COMPARISON
SQL_SQ_EXISTS
SQL_SQ_IN
SQL_SQ_QUATIFIED

The SQL_SQ_CORRELATED_SUBQUERIES bitmask indicates that all predicates that support subqueries support correlated subqueries.

See also ODBC_GET_INFO@

SQL_SYSTEM_FUNCTIONS

Description A 32-bit bitmask enumerating the scalar system functions supported by the driver and associated data source. The following bitmasks are used to determine which system functions are supported:

SQL_FN_SYS_DBNAME
SQL_FN_SYS_IFNULL
SQL_FN_SYS_USERNAME

See also ODBC_GET_INFO@

SQL_TABLE_TERM

Description A character string with the data source vendor's name for a table; for example, "table" or "file".

See also ODBC_GET_INFO@

SQL_TIMEDATE_ADD_INTERVALS

Description A 32-bit bitmask enumerating the timestamp intervals supported by the driver and associated data source for the TIMESTAMPADD scalar function. The following bitmasks are used to determine which intervals are supported:

SQL_FN_TSI_FRAC_SECOND	SQL_FN_TSI_SECOND
SQL_FN_TSI_MINUTE	SQL_FN_TSI_HOUR
SQL_FN_TSI_DAY	SQL_FN_TSI_WEEK
SQL_FN_TSI_MONTH	SQL_FN_TSI_QUARTER
SQL_FN_TSI_YEAR	

See also ODBC_GET_INFO@

SQL_TIMEDATE_DIFF_INTERVALS

Description A 32-bit bitmask enumerating the timestamp intervals supported by the driver and associated data source for the TIMESTAMPDIFF scalar function. The following bitmasks are used to determine which intervals are supported:

SQL_FN_TSI_FRAC_SECOND	SQL_FN_TSI_SECOND
SQL_FN_TSI_MINUTE	SQL_FN_TSI_HOUR
SQL_FN_TSI_DAY	SQL_FN_TSI_WEEK
SQL_FN_TSI_MONTH	SQL_FN_TSI_QUARTER
SQL_FN_TSI_YEAR	

See also ODBC_GET_INFO@

SQL_TIMEDATE_FUNCTIONS

Description A 32-bit bitmask enumerating the scalar date and time functions supported by the driver and associated data source. The following bitmasks are used to determine which date and time functions are supported:

SQL_FN_TD_CURDATE SQL_FN_TD_CURTIME
SQL_FN_TD_DAYNAME SQL_FN_TD_DAYOFMONTH
SQL_FN_TD_DAYOFWEEK SQL_FN_TD_DAYOFYEAR
SQL_FN_TD_HOUR SQL_FN_TD_MINUTE
SQL_FN_TD_MONTH SQL_FN_TD_MONTHNAME
SQL_FN_TD_NOW SQL_FN_TD_QUARTER
SQL_FN_TD_SECOND SQL_FN_TD_TIMESTAMPADD
SQL_FN_TD_TIMESTAMPDIFF
SQL_FN_TD_WEEK
SQL_FN_TD_YEAR

See also ODBC_GET_INFO@

SQL_TXN_CAPABLE

Description An integer value describing the transaction supported in the driver or data source:

SQL_TC_NONE means transactions are not supported.

SQL_TC_DML means transactions can only contain Data Manipulation Language (DML) statements (**SELECT**, **INSERT**, **UPDATE**, **DELETE**). Data Definition Language statements encountered in a transaction cause an error.

SQL_TC_DDL_COMMIT means transactions can only contain Data Manipulation Language (DML) statements (**SELECT**, **INSERT**, **UPDATE**, **DELETE**). Data Definition Language statements, such as **CREATE_TABLE** and **DROP_INDEX**, encountered in a transaction cause the transaction to be committed.

SQL_TC_DDL_IGNORE means transactions can only contain Data Manipulation Language (DML) statements. Data Definition Language statements encountered in a transaction are ignored.

SQL_TC_DDL_ALL means transactions can contain Data Manipulation Language (DML) statements and Data Definition Language (DDL) statements in any order.

See also ODBC_GET_INFO@

SQL_TXN_ISOLATION_OPTION

Description A 32-bit bitmask enumerating the transaction isolation levels available from the driver or data source. The following bitmasks are used in conjunction with the flag to determine which options are supported:

SQL_TXN_READ_UNCOMMITTED

SQL_TXN_READ_COMMITTED

SQL_TXN_REPEATABLE_READ

SQL_TXN_SERIALIZABLE

SQL_TXN_VERSIONING

For descriptions of these isolation levels, see the description of SQL_DEFAULT_TXN_ISOLATION.

See also ODBC_GET_INFO@

SQL_UNION

Description A 32-bit bitmask enumerating support for the **UNION** clause:

SQL_U_UNION means the data source supported the **UNION** clause.

SQL_U_UNION_ALL means the data source supports the ALL keyword in the **UNION** clause. (ODBC_GET_INFO@ returns both SQL_U_UNION and SQL_U_UNION_ALL in this case.

See also ODBC_GET_INFO@

SQL_USER_NAME

Description A character string with the name used in a particular database, which can be different than login name.

See also ODBC_GET_INFO@

SQL_ASYNC_ENABLE

Description An integer that specifies whether a function called with the specified statement handle is executed asynchronously.

SQL_ASYNC_ENABLE_OFF means the statement is not called asynchronously.

SQL_ASYNC_ENABLE_ON means the statement is called asynchronously.

Once a function has been called asynchronously, only four other functions may be called using that statement handle or database handle:

- The original function
- ODBC_ALLOC_STMT@
- ODBC_CANCEL@
- ODBC_GET_FUNCTIONS@

Any attempt to call another macro on that statement handle fails until the executing statement returns a status other than SQL_STILL_EXECUTING. Function called on other statement handles work normally. The following functions can be executed asynchronously:

ODBC_COL_ATTRIBUTES@	ODBC_NUM_PARAMS@
ODBC_COLUMN_PRIVILEGES@	ODBC_NUM_RESULT_COLS@
ODBC_COLUMNS@	ODBC_PARAM_DATA@
ODBC_DESCRIBE_COLS@	ODBC_PREPARE@
ODBC_DESCRIBE_PARAM@	ODBC_PRIMARY_KEYS@
ODBC_EXEC_DIRECT@	
ODBC_PROCEDURE_COLUMNS@	ODBC_EXTENDED_FETCH@
ODBC_PUT_DATA@	ODBC_FETCH@
ODBC_SET_POS@	ODBC_FOREIGN_KEYS@
ODBC_SPECIAL_COLUMNS@	
ODBC_GET_DATA@	ODBC_STATISTICS@
ODBC_GET_TYPE_INFO@	ODBC_TABLE_PRIVILEGES@
ODBC_MORE_RESULTS@	ODBC_TABLES@

SQL_BIND_TYPE

Description The binding orientation to be used when ODBC_EXTENDED_FETCH@ is called on the associated statement handle. This value is SQL_BIND_BY_COLUMN.

The ELF ODBC interface supports only column-wise binding. Row-wise binding is not supported.

SQL_BIND_TYPE

Description An integer indicating cursor concurrency:

SQL_CONCUR_READ_ONLY indicates that the cursor is Read-only. No updates are allowed.

SQL_CONCUR_LOCK indicates that the cursor uses the lowest level of locking sufficient to ensure that the row can be updated.

SQL_CONCUR_ROWVER indicates that the cursor uses optimistic concurrency controls comparing row versions, such as SQLBase ROWID, or Sybase TIMESTAMP.

SQL_CONCUR_VALUES indicates that the cursor uses optimistic concurrency controls comparing values.

The default value is SQL_CONCUR_READ_ONLY. This option cannot be specified for an open cursor. It can also be set through the concurrency argument in ODBC_SET_SCROLL_OPTIONS@.

If the specified concurrency is not supported by the data source, the driver substitutes a different concurrency and return SQLSTATE 01S02 (Option value changed). For SQL_CONCUR_VALUES, the driver substitutes SQL_CONCUR_ROWVER, and vice versa. For SQL_CONCUR_LOCK the driver substitutes, in order, SQL_CONCUR_ROWVER or SQL_CONCUR_VALUES.

SQL_CURSOR_TYPE

Description A 32-bit integer value that specifies the cursor type:

SQL_SO_FORWARD_ONLY means the cursor only scrolls forward.

SQL_SO_STATIC means the data in the result set is static.

SQL_SO_KEYSET_DRIVEN means the driver saves and uses the keys for the number of rows specified in the SQL_KEYSET_SIZE statement option.

SQL_SO_DYNAMIC means the driver keeps the keys for every row in the rowset.

The default value is SQL_SO_FORWARD_ONLY. This option cannot be specified for an open cursor and can also be set through the crowKeyset argument in ODBC_SET_SCROLL_OPTIONS@. If the specified cursor type is not supported by the data source, the driver substitutes a different cursor type and returns SQLSTATE 01S02 (Option value changed.) For a mixed or dynamic cursor, the driver substitutes, in order, a keyset-driven or static cursor. For a keyset-driven cursor, the driver substitutes a static cursor.

SQL_KEYSET_SIZE

Description A 32-bit integer value that specifies the number of rows in the keyset for a keyset-driven cursor. If the keyset size is 0 (the default), the cursor is fully keyset-driven. If the keyset size if greater than 0, the cursor is mixed (keyset-driven within the keyset, and dynamic outside of the keyset). The default keyset size is 0.

If the specified size exceeds the maximum keyset size, the driver substitutes that size and returns SQLSTATE 01S02 (Option value changed).

ODBC_EXTENDED_FETCH@ returns an error if the keyset size is greater than 0, ans less than the rowset size.

SQL_MAX_LENGTH

Description An integer value that specifies the maximum amount of data that the driver returns from a character or binary column. If vParam is less than the length of the available data, ODBC_FETCH@ or ODBC_GET_DATA@ truncates the data and returns SQL_SUCCESS. If *vParam* is zero (the default), the driver attempts to return all available data.

If the specified length is less than the minimum amount of data that the data source can return, (the minimum is 254 bytes on many data sources), or greater than the maximum amount of data that the data source can return, the driver substitutes that value and returns SQLSTATE 01s02 (Option value changed).

The option is intended to reduce network traffic and should only be supported when the data source (as opposed to the driver) in a multiple-tier driver can implement it. To truncate data, an application should specify the maximum buffer length in the *MaxLength* argument in ODBC_BIND_COL@ or ODBC_GET_DATA@.

SQL_MAX_ROWS

Description A 32-bit integer value corresponding to the maximum number of rows to return to the application for a **SELECT** statement. If *Param* equals 0, then the driver returns all rows.

This option is intended to reduce network traffic. It is applied when the result set is created and limits the result set to *Param* rows.

If the specified number of rows exceeds the number of rows that can be returned by the data source, the driver substitutes that value and return SQLSTATE 01s02 (Option value changed).

SQL_NOSCAN

Description An integer value that specifies whether or not the driver scans SQL strings for escape clauses:

SQL_NOSCAN_OFF means the driver scans SQL strings for escape clauses. This is the default.

SQL_NOSCAN_ON means the driver does not scan SQL strings for escape clauses. Instead, the driver sends the statement directly to the data source.

SQL_QUERY_TIMEOUT

Description An integer value corresponding to the number of seconds to wait for an SQL statement to execute before returning to the application. If param equals 0, then there is no timeout.

If the specified timeout exceeds the maximum timeout in the data source or is smaller than the minimum timeout, the driver substitutes that value and returns SQLSTATE 01s02 (Option value changed).

Note that the application need not call ODBC_FREE_STMT@ with the SQL_CLOSE option to reuse the statement handle if a **SELECT** statement timed out.

SQL_RETRIEVE_DATA

Description A 32-bit integer value:

SQL_RD_ON means ODBC_EXTENDED_FETCH@ retrieves the data after it positions the cursor to the specified location. This is the default.

SQL_RD_OFF means ODBC_EXTENDED_FETCH@ does not retrieve data after it positions the cursor.

By setting SQL_RETRIEVE_DATA to SQL_RD_OFF, an application can verify if a row exists or retrieve a bookmark for the row without incurring the overhead of retrieving rows.

SQL_ROWSET_SIZE

Description An integer value that specifies the number of rows in the rowset. This is the number of rows returned by each call to ODBC_EXTENDED_FETCH@. The default value is 1.

If the specified rowset size exceeds the maximum rowset size supported by the data source, the driver substitutes that value and returns SQLSTATE 01s02 (Option value changed).

SQL_SIMULATE_CURSOR

Description A 32-bit integer value that specifies whether drivers that simulate positioned update and delete statements guarantee that such statements affect only one single row.

To simulate positioned update and delete statements, most drivers contruct a searched **UPDATE** or **DELETE** statement containing a **WHERE** clause that specifies the value of each column in the current row. Unless these columns comprise a unique key, such a statement may affect more than one row.

To guarantee that such statements affect only one row, the driver determines the columns in a unique key and adds these columns to the result set. If an application guarantees that the columns in the result set comprise a unique key, the driver is not required to do so. This may reduce execution time.

SQL_NC_NON_UNIQUE means the driver does not guarantee that simulated positioned update or delete statement will affect only one row. It is, therefore, the application's responsibility to do so. If a statement affects more than one row,

ODBC_EXECUTE@ or ODBC_EXECUTE_DIRECT@ returns SQLSTATE 01000 (General warning).

SQL_NC_TRY_UNIQUE means the driver attempts to guarantee that simulated positioned update or delete statement will affect only one row. The driver always executes such statements, even if they might affect more than one row, such as when there is no unique key. If a statement affects more than one row, ODBC_EXECUTE@ or ODBC_EXECUTE_DIRECT@ returns SQLSTATE 01000 (General warning).

SQL_SC_UNIQUE means the driver guarantees that simulated positioned update or delete statements affect only one row. If the driver cannot guarantee this for a given statement, ODBC_EXECUTE_DIRECT@ or ODBC_PREPARE@ return an error.

If the specified cursor simulation type is not supported by the data source, the driver substitutes a different simulation type and returns SQLSTATE 01s02 (Option value changed). For SQL_SC_UNIQUE, the driver substitutes, in order, SQL_NC_TRY_UNIQUE and SQL_NC_NON_UNIQUE. For SQL_NC_TRY_UNIQUE, the driver substitutes SQL_NC_NON_UNIQUE.

If the driver does not simulate positioned update and delete statements, it returns SQLSTATE S1C00 (Driver not capable).

SQL_USE_BOOKMARKS

Description An integer value that specifies whether or not an application will use bookmarks with a cursor:

SQL_UB_OFF = Off (the default)

SQL_UB_ON = On

Bookmarks are not supported by the ELF ODBC interface.

SQL@

Executes an SQL statement

Format info = SQL@(channel, sqlStmt)

Arguments

channel	The channel id returned by **SQL_CONNECT@**.
sqlStmt	The command being executed; sql_stmt can be a string or an array of strings.

Description Executes the passed statement directly. No explicit *prepare* statements or cursor calls need to be used.

If sqlStmt is not a query statement, ELF SQL performs the following steps:

1. Prepares the statement.
2. Executes the statement.
3. Unprepares the statement.

Any data that would be returned if sqlStmt is returned.

If the statement is a query statement, the following sequence of operations occurs:

1. Prepares the statement.
2. Declares the cursor.
3. Opens the cursor.
4. Fetches all of the data.
5. Closes the cursor.
6. Unprepares the statement.

While it is easy to invoke SQL statements using the SQL@ function, it can greatly increase your overhead because of the added expense of repeatedly preparing the statement.

The information that is returned by this macro varies and is dependent upon the statement that is executed.

See also SQL_CHANNEL_MASTER@, SQL_CLOSE_CURSOR@, SQL_CONNECT@, SQL_DECLARE_CURSOR@, SQL_DESCRIBE@, SQL_DISCONNECT@, SQL_EXECUTE_STMT@, SQL_FETCH_CURSOR@, SQL_LOGGING@, SQL_OPEN_CURSOR@, SQL_PREPARE@, SQL_PREPARE_FOR_UPDATE@, SQL_PROCEDURE@, SQL_READ_SQLSTR_FROM_FILE@, SQL_SELECT@ and SQL_UNPREPARE@.

SQL_CHANNEL_MASTER@

Sets the channel existence state when the SQL task goes away

Format SQL_CHANNEL_MASTER@(format sql_channel@ channel, taskID)

Arguments

channel	The channel id returned by SQL_CONNECT@.
taskID	The ID of an executing task.

Description Indicates that the channel is disconnected when a task stops executing. The exception is when you set taskID to 0, which couples it to no task. In this case, the channel will remain even if the task goes away. Normally, you would only use this macro when you want a channel to a running server process to remain open at all times.

See also SQL@, SQL_CLOSE_CURSOR@, SQL_CONNECT@, SQL_DECLARE_CURSOR@, SQL_DESCRIBE@, SQL_DISCONNECT@, SQL_EXECUTE_STMT@, SQL_FETCH_CURSOR@, SQL_LOGGING@, SQL_OPEN_CURSOR@, SQL_PREPARE@, SQL_PREPARE_FOR_UPDATE@, SQL_PROCEDURE@, SQL_READ_SQLSTR_FROM_FILE@, SQL_SELECT@ and SQL_UNPREPARE@.

SQL_CLOSE_CURSOR@

Closes a cursor

Format SQL_CLOSE_CURSOR@(channel, cursorNum)

Arguments

channel	The channel id returned by SQL_CONNECT@.
cursorNum	The cursor number returned by SQL_DECLARE_CURSOR@

Description Releases the resources allocated for a cursor. After a cursor is closed, DML operations using the cursor can no longer be performed.

See also SQL@, SQL_CHANNEL_MASTER@, SQL_CONNECT@, SQL_DECLARE_CURSOR@, SQL_DESCRIBE@, SQL_DISCONNECT@, SQL_EXECUTE_STMT@, SQL_FETCH_CURSOR@, SQL_LOGGING@, SQL_OPEN_CURSOR@, SQL_PREPARE@, SQL_PREPARE_FOR_UPDATE@, SQL_PROCEDURE@, SQL_READ_SQLSTR_FROM_FILE@, SQL_SELECT@ and SQL_UNPREPARE@.

SQL_CONNECT@

Connects to a database

Format channel = SQL_CONNECT@(vendor, username, password, database, host, server, routing[, format sql_cursor_properties cursorProps[, connectInfo])

Arguments

vendor	The string name of the DBMS: "Oracle", "Sybase", "Informix", "ODBC"
username	Your user name--*Oracle* and *Sybase* only; NULL otherwise.
password	Your password--*Oracle* and *Sybase* only; NULL otherwise.
database	The database name. If you are connecting through ODBC, this is the vendor name ("Oracle", "Sybase", "Informix").
host	If you are using Axnet, this is the host where the gateway is running.
server	Sybase only; otherwise, NULL.

routing (Oracle SQL*Net Version 1 only) If you are not using Axnet, this is the name of the host you are trying to reach.

If you are using DECnet, the routing name must begin with d:.

cursorProps The properties for retrieving and presenting the database information, as follows:

```
format sql_cursor_properties
    fetch_size,    'Number of records/fetch: default is 50
    trim_strings,  'Trim (remove whitespace) strings
                   'SQL_TRIM_BOTH_
                   'SQL_TRIM_BEGIN_
                   'SQL_TRIM_END_
                   'SQL_TRIM_NONE_
    transpose_data  'Transpose data? Default is FALSE
    nums_as_nums  'Number as numbers: Default is FALSE (that is, show
                   as strings)
    binary_size_limit,
                   'max size of binary data item fetched
    timeout,       'timeout in seconds, zero for infinite, null for default
    max_rows       'maximum number of rows (passed to DBMS server), null
                   for default
    dates_as_dates
                   ' If TRUE, date fields are returned as
                   ' date arrays. If FALSE, date fields are
                   ' returned as strings.
```

info (Builder only) An array containing the following connection information:

```
info[0]    noTransactionsFlag
info[1]    notAnsiFlag
info[2]    notSQLFlag
```

Description Establishes a connection to a database and returns a channel number. This channel number will be used in almost every ELF SQL macro. This macro always starts a gateway running.

The channel will disappear when the creating task goes away, unless SQL_CHANNEL_MASTER@ or SQL_DISCONNECT@ has been called.

If you are connecting to ODBC, no channel number is returned. Instead, SQL_CONNECT@ returns a Builder object string. This is an opaque data structure that is used in place of the channel for subsequent SQL macro calls.

To be sure that your connection to the ODBC database was successful, use the IS_OBJECT@ macro to test the value returned from the SQL_CHANNEL@ call. If the test is TRUE, the connection was successfully made. If the test is FALSE, no connection was established.

See also SQL@, SQL_CHANNEL_MASTER@, SQL_CLOSE_CURSOR@, SQL_DECLARE_CURSOR@, SQL_DESCRIBE@, SQL_DISCONNECT@, SQL_EXECUTE_STMT@, SQL_FETCH_CURSOR@, SQL_LOGGING@, SQL_OPEN_CURSOR@, SQL_PREPARE@, SQL_PREPARE_FOR_UPDATE@, SQL_PROCEDURE@, SQL_READ_SQLSTR_FROM_FILE@, SQL_SELECT@ and SQL_UNPREPARE@.

SQL_DECLARE_CURSOR@

Creates a cursor

Format cursorNum = SQL_DECLARE_CURSOR@(channel, stmtNum)

Arguments

channel	The channel id returned by SQL_CONNECT@.
stmtNum	The statement number returned by SQL_PREPARE@.

Description Declares a cursor for an already PREPAREd statement. This macro returns the cursor id (cursorNum). Restated, this macro associates a cursor with a query. Cursors are used when retrieving data and when performing positioned forms of the DELETE and UPDATE statements.

The SQL_DECLARE_CURSOR@ macro must be executed before each execution of the OPEN statement for the same cursor.

See also SQL@, SQL_CHANNEL_MASTER@, SQL_CLOSE_CURSOR@, SQL_CONNECT@, SQL_DESCRIBE@, SQL_DISCONNECT@, SQL_EXECUTE_STMT@, SQL_FETCH_CURSOR@, SQL_LOGGING@, SQL_OPEN_CURSOR@, SQL_PREPARE@, SQL_PREPARE_FOR_UPDATE@, SQL_PROCEDURE@, SQL_READ_SQLSTR_FROM_FILE@, SQL_SELECT@ and SQL_UNPREPARE@.

SQL_DESCRIBE@

Implements Oracles' Describe Statement

Format infoArray = SQL_DESCRIBE@(channel, stmtNum)

Arguments

channel	The channel id returned by SQL_CONNECT@.
stmtNum	The statement upon which a *prepare* operation was run.

Description Returns two arrays for a prepared statement: an array of column names and column types (Oracle only).

See also SQL@, SQL_CHANNEL_MASTER@, SQL_CLOSE_CURSOR@, SQL_CONNECT@, SQL_DECLARE_CURSOR@, SQL_DISCONNECT@, SQL_EXECUTE_STMT@, SQL_FETCH_CURSOR@, SQL_LOGGING@, SQL_OPEN_CURSOR@, SQL_PREPARE@, SQL_PREPARE_FOR_UPDATE@, SQL_PROCEDURE@, SQL_READ_SQLSTR_FROM_FILE@, SQL_SELECT@ and SQL_UNPREPARE@.

SQL_DISCONNECT@

Removes a connection to a database

Format SQL_DISCONNECT@(channel)

Arguments

channel	The channel id returned by SQL_CONNECT@.

Description Closes communication to the gateway and terminates the gateway program. It also closes the connection to the DBMS.

See also SQL@, SQL_CHANNEL_MASTER@, SQL_CLOSE_CURSOR@, SQL_CONNECT@, SQL_DECLARE_CURSOR@, SQL_DESCRIBE@, SQL_EXECUTE_STMT@, SQL_FETCH_CURSOR@, SQL_LOGGING@, SQL_OPEN_CURSOR@, SQL_PREPARE@, SQL_PREPARE_FOR_UPDATE@, SQL_PROCEDURE@, SQL_READ_SQLSTR_FROM_FILE@, SQL_SELECT@ and SQL_UNPREPARE@.

SQL_EXECUTE_STMT@

Executes a prepared statement

Format [result =] SQL_EXECUTE_STMT@(channel, stmtNum[, bindValues])

Arguments

channel	The channel id returned by SQL_CONNECT@.
stmtNum	The statement number returned by SQL_PREPARE@.
bindValues	Values to be used in the non-query statement. These values can be an array or a two-dimensional array. Some DBMS engines allow multiple updates or inserts. These use a two-dimensional array. If your DBMS supports multiple updates or inserts, SQL ELF determines what you are trying to do based on the number of dimensions in bindValues. A vector means a single record is being added or updated; a two-dimensional array means multiple insertions or updates are being made.

Description Executes a prepared statement; prepares non-query statement with bind values. A prepared statement can be executed one or more times with different host variables by repeatedly calling this macro.

Host variables can only be used as input arguments to the SQL statement being executed. All SQL statements except SELECT can be prepared and executed in this manner.

The number of records updated is returned if this is an update statement.

Here is an example:

```
stmtNum = SQL_PREPARE@(channel,
                       "UPDATE emp SET sal = :newsal")
bindValue[0] = 3000
table = SQL_EXECUTE_STMT@(channel, stmtNum,
                          bindValues)
flag = SQL_UNPREPARE@(channel, stmtNum)
```

See also SQL@, SQL_CHANNEL_MASTER@, SQL_CLOSE_CURSOR@, SQL_CONNECT@, SQL_DECLARE_CURSOR@, SQL_DESCRIBE@, SQL_DISCONNECT@, SQL_FETCH_CURSOR@, SQL_LOGGING@, SQL_OPEN_CURSOR@, SQL_PREPARE@, SQL_PREPARE_FOR_UPDATE@, SQL_PROCEDURE@, SQL_READ_SQLSTR_FROM_FILE@, SQL_SELECT@ and SQL_UNPREPARE@.

SQL_FETCH_CURSOR@

Retrieves information

Format

```
table = SQL_FETCH_CURSOR@(channel, cursorNum
                          [, numRecsRequested[, headerFlag]])
```

Arguments

channel	The channel id returned by SQL_CONNECT@.
cursorNum	The cursor number returned by SQL_DECLARE_CURSOR@.
numRecsRequested	The maximum number of records to be retrieved. NULL means 1 record. There is no limit on the number of record you can retrieve in a request.
headerFlag	A Boolean value which if set to TRUE indicates that the first row will contain the column's headers.

Description Fetches the *select* list into a *select* cursor. Executing this macro also moves the position of the cursor to the next row of the active set and fetches the column values of the current row into table.

The cursor must be open for the FETCH operation to succeed.

If numRecsRequested is NULL, only one record is retrieved. This retrieved information is returned as a vector. If however, you specify that 1 record should be retrieved, a two-dimensional array is returned. This *table* has two rows: the column information and the table data. The column information is in col_info@ format, which is as follows:

```
format col_info@    'header info on sql_select when
                    ' headerFlag
      names,        'array of heading names
      types         'array of data types
```

If you specify one record and do not want headers, you still receive a two-dimensional array. However, the array size of the first dimension is one, not two.

Because all information is returned as strings, you need to use the types field to determine if you can manipulate the data and, if you can, how you should manipulate it. For example, an employee id number could be all numeric; however, you would treat it as a string; not as a number. (It doesn't make sense to subtract 3 from an employee number.)

If you try to fetch records after all records are fetched, table is set to NULL. If the number or records retrieved is less than numRecsRequested (because the records do not exist in the database) no error occurs. It is up to you to test for this condition.

The following uses a FETCH to retrieve 10 records.

```
stmtNum = SQL_PREPARE@
                (channel,"SELECT * FROM parts")
cursorNum = SQL_DECLARE_CURSOR@
                (channel, stmtNum)
SQL_OPEN_CURSOR@(channel, cursorNum)
table = SQL_FETCH_CURSOR@
                (channel, cursorNum, 10)
flag = SQL_CLOSE_CURSOR@(channel, cursorNum)
flag = SQL_UNPREPARE@(channel, stmtNum)
```

See also SQL@, SQL_CHANNEL_MASTER@, SQL_CLOSE_CURSOR@, SQL_CONNECT@, SQL_DECLARE_CURSOR@, SQL_DESCRIBE@, SQL_DISCONNECT@, SQL_EXECUTE_STMT@, SQL_LOGGING@, SQL_OPEN_CURSOR@, SQL_PREPARE@, SQL_PREPARE_FOR_UPDATE@, SQL_PROCEDURE@, SQL_READ_SQLSTR_FROM_FILE@, SQL_SELECT@ and SQL_UNPREPARE@.

SQL_LOGGING@

Writes state information

Format SQL_LOGGING@(channel, fileOrFlag)

Arguments

channel	The channel id returned by **SQL_CONNECT@**.
fileOrFlag	The name of the file to which log information is written, or a Boolean value. If the value is set to FALSE, no logging occurs. If the value is set to TRUE, log information is written to STDOUT. If the value is a file name, then log information is written to the file.

Description Tells the gateway to write debugging information to STDOUT or a file. If you are using axnet, STDOUT goes to the window from which axnet was started. As axnet is usually started when the computer is booted, this information appears in the console window.

See also **SQL@**, **SQL_CHANNEL_MASTER@**, **SQL_CLOSE_CURSOR@**, **SQL_CONNECT@**, **SQL_DECLARE_CURSOR@**, **SQL_DESCRIBE@**, **SQL_DISCONNECT@**, **SQL_EXECUTE_STMT@**, **SQL_FETCH_CURSOR@**, **SQL_OPEN_CURSOR@**, **SQL_PREPARE@**, **SQL_PREPARE_FOR_UPDATE@**, **SQL_PROCEDURE@**, **SQL_READ_SQLSTR_FROM_FILE@**, **SQL_SELECT@** and **SQL_UNPREPARE@**.

SQL_OPEN_CURSOR@

Opens a created cursor

Format SQL_OPEN_CURSOR@(channel, cursorNum[, bindValues[, format sql_cursor_properties cursor_properties]])

Arguments

channel	The channel id returned by **SQL_CONNECT@**.
cursorNum	The cursor number returned by **SQL_DECLARE_CURSOR@**.
bindValues	Values to be added to the select statement.
cursor_properties	The properties for retrieving and presenting the database information.

Description Stores bind variables and the select list and opens the cursor. This macro does most of the cursor-related operations.

Opening a cursor tells ELF to execute the query specified in the DECLARE CURSOR statement and create the rows that satisfy the select statement. The collection of retrieved row is called the active set.

This statement positions the cursor just before the first row of the active set.

After the cursor is opened, the active set does not change and the bind variables are not re-examined.

If the statement is not a SELECT, the statement is also executed and its results are returned.

The definition of sql_cursor_properties@ is as follows:

```
format sql_cursor_properties@
        fetch_size,     'Number for records/fetch. The default is 50.
        trim_strings,   'Trim strings. The default is to trim left and right.
        transpose_data,
                        'Boolean: default is FALSE
        nums_as_nums,
                        'Numbers as numbers: default is FALSE; numbers are treated as
                        strings
        binary_size_limit,
                        'Maximum size of binary data item fetched.
        timeout,        'Timeout in seconds. Zero for infinite; NULL for the default
        max_rows        'Maximum number or rows passed to the DBMS server; NULL for
                        default.
```

See also SQL@, SQL_CHANNEL_MASTER@, SQL_CLOSE_CURSOR@, SQL_CONNECT@, SQL_DECLARE_CURSOR@, SQL_DESCRIBE@, SQL_DISCONNECT@, SQL_EXECUTE_STMT@, SQL_FETCH_CURSOR@, SQL_LOGGING@, SQL_PREPARE@, SQL_PREPARE_FOR_UPDATE@, SQL_PROCEDURE@, SQL_READ_SQLSTR_FROM_FILE@, SQL_SELECT@ and SQL_UNPREPARE@.

SQL_PREPARE@

Prepares a statement

Format stmtNum = SQL_PREPARE@(channel, sqlStmt)

Arguments

channel	The channel id returned by SQL_CONNECT@.
sqlStmt	The statement being prepared; this can be a string or an array of strings.

Description Prepares and parses a SQL statement. This macro also allocates memory for the bind values and the data to be returned from your DBMS.

See also SQL@, SQL_CHANNEL_MASTER@, SQL_CLOSE_CURSOR@, SQL_CONNECT@, SQL_DECLARE_CURSOR@, SQL_DESCRIBE@, SQL_DISCONNECT@, SQL_EXECUTE_STMT@, SQL_FETCH_CURSOR@, SQL_LOGGING@, SQL_OPEN_CURSOR@, SQL_PREPARE_FOR_UPDATE@, SQL_PROCEDURE@, SQL_READ_SQLSTR_FROM_FILE@, SQL_SELECT@ and SQL_UNPREPARE@.

SQL_PREPARE_FOR_UPDATE@

Prepares a statement and allows record locking

Format stmtNum = SQL_PREPARE_FOR_UPDATE@(channel, selectStmt[, waitForLockFlag])

Arguments

channel The channel id returned by SQL_CONNECT@.

selectStmt The statement being prepared; this can be a string or an array of strings.

waitForLockFlag
A Boolean value that tells ELF to wait if another process has the record locked.

Description Prepares and parses an SQL select statement. When a select is executed that has been prepared for update, the selected record is also locked. This way, you can change the record after viewing it. The normal sequence of operations is as follows:

```
PREPARE (select statement)
DECLARE
OPEN
FETCH
    PREPARE (update statement)
    EXECUTE
    UNPREPARE (update statement)
CLOSE
UNPREPARE (select)
```

Normally, only one record at a time is locked. However, the scope of the lock can be greater than one record. For example, some DBMS engines automatically round the locking to the nearest page boundary.

See also SQL@, SQL_CHANNEL_MASTER@, SQL_CLOSE_CURSOR@, SQL_CONNECT@, SQL_DECLARE_CURSOR@, SQL_DESCRIBE@, SQL_DISCONNECT@, SQL_EXECUTE_STMT@, SQL_FETCH_CURSOR@, SQL_LOGGING@, SQL_OPEN_CURSOR@, SQL_PREPARE@, SQL_PROCEDURE@, SQL_READ_SQLSTR_FROM_FILE@, SQL_SELECT@ and SQL_UNPREPARE@.

SQL_PROCEDURE@

Executes a stored database procedure

Format SQL_PROCEDURE@(channel, procName[, parameters])

Arguments

channel	The channel id returned by SQL_CONNECT@.
procName	The name of the stored procedure
parameters	Any parameters needed by procName.

Description Executes a stored database procedure (providing that your database supports stored procedures). The database procedure will be created and stored using the tools and language of the database. This macro invokes the procedure from within ELF.

For example, assume that you have created the following Sybase stored procedure:

```
create procedure test1 @time datetime as
        select * from syslogins where accdate >= @time
```

Here is a macro that will execute this procedure:

```
include "dbase_.am"
MACRO Sql_Procedure_Test
        var result, channel, arr, procname
        var format sql_procedure_params@ param
        var format sql_procedure_result@ results

        channel = SQL_CONNECT@("Sybase", "sa", "", "master", "me")

        param.data = "Dec 16 1993"
        param.type = SYB#DATETIME_
        param.output = FALSE
        procname = "test1"

        arr[0] = param
        results = SQL_PROCEDURE@(channel,procname,arr)
        DUMP_ARRAY@(results)
ENDMACRO
```

See also SQL@, SQL_CHANNEL_MASTER@, SQL_CLOSE_CURSOR@, SQL_CONNECT@, SQL_DECLARE_CURSOR@, SQL_DESCRIBE@, SQL_DISCONNECT@, SQL_EXECUTE_STMT@, SQL_FETCH_CURSOR@, SQL_LOGGING@, SQL_OPEN_CURSOR@, SQL_PREPARE@, SQL_PREPARE_FOR_UPDATE@, SQL_READ_SQLSTR_FROM_FILE@, SQL_SELECT@ and SQL_UNPREPARE@.

SQL_READ_SQLSTR_FROM_FILE@

Reads the SQL string from a query file

Format SQL_READ_SQLSTR_FROM_FILE@(filename)

Arguments filename An Applixware Data file.

Description Reads an Applixware Data file and returns the SQL string stored within it.

SQL_SELECT@

Executes an SQL SELECT

Format table = sql_select@(channel, queryStmt[, maxRecords[, headersFlag]])

Arguments

channel	The channel id returned by SQL_CONNECT@.
queryStmt	The SELECT statement to execute.
maxRecords	The maximum number of records to be retrieved. There is no limit on the number of record you can retrieve in a request.
headersFlag	A Boolean value, which if set to TRUE indicates that the first retrieved row in table will contain the column's headers.

Description Retrieves information from your database. Because this statement will prepare, declare, and fetch information, it is inherently less efficient if you will be performing a series of interactions.

This macro returns a two-dimensional array of strings. The first array dimension is the row; the second array dimension is one or more columns of information in the row.

See also SQL@, SQL_CHANNEL_MASTER@, SQL_CLOSE_CURSOR@, SQL_CONNECT@, SQL_DECLARE_CURSOR@, SQL_DESCRIBE@, SQL_DISCONNECT@, SQL_EXECUTE_STMT@, SQL_FETCH_CURSOR@, SQL_LOGGING@, SQL_OPEN_CURSOR@, SQL_PREPARE@, SQL_PREPARE_FOR_UPDATE@, SQL_PROCEDURE@, and SQL_UNPREPARE@.

SQL_UNPREPARE@

Releases memory resources used by a statement

Format flag = SQL_UNPREPARE@(channel, stmtNum)

Arguments

channel	The channel id returned by SQL_CONNECT@.
stmtNum	A statement number returned by SQL_PREPARE@.

Description Releases the resources acquired by a prepared statement.

See also SQL@, SQL_CHANNEL_MASTER@, SQL_CLOSE_CURSOR@, SQL_CONNECT@, SQL_DECLARE_CURSOR@, SQL_DESCRIBE@, SQL_DISCONNECT@, SQL_EXECUTE_STMT@, SQL_FETCH_CURSOR@, SQL_LOGGING@, SQL_OPEN_CURSOR@, SQL_PREPARE@, SQL_PREPARE_FOR_UPDATE@, SQL_PROCEDURE@, SQL_READ_SQLSTR_FROM_FILE@, and SQL_SELECT@.

ELF SQL

ELF SQL allows you to execute SQL statements with one of the ELF SQL macros. For example:

- sql_select@(channel,"select * from parts")

While you can build applications that perform any kind of database activity, the most common applications will be those that:

- Create GUI applications that allow end users to retrieve, add, modify, and delete information. Use ELF SQL to support DML activities.
- Extract information from databases and use Applixware applications to display or manipulate this information.

For more information, select one of the following topics:

SQL Macro Command Summary

SQL Macro Command Summary

Here are the ELF SQL macros:

SQL@
: Executes an SQL statement

SQL_CLOSE_CURSOR@
: Closes a cursor

SQL_CONNECT@
: Connects to a database

SQL_DECLARE_CURSOR@
: Creates a cursor

SQL_DISCONNECT@
Removes a connection to a database
SQL_EXECUTE_STMT@
Executes a prepared statement
SQL_FETCH_CURSOR@
Retrieves information
SQL_LOGGING@
Writes state information
SQL_OPEN_CURSOR@
Opens a created cursor
SQL_PREPARE@
Prepares a statement
SQL_PREPARE_FOR_UPDATE@
Prepares a statement and allows record locking
SQL_SELECT@
Retrieves information
SQL_UNPREPARE@
Releases memory resources used by a statement

For more information, select one of the following topics:

ELF SQL