

ODBC Driver for OMNIS Data Files

OMNIS Software

August 1998

The software this document describes is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement. Names of persons, corporations, or products used in the tutorials and examples of this manual are fictitious. No part of this publication may be reproduced, transmitted, stored in a retrieval system or translated into any language in any form by any means without the written permission of OMNIS Software.

© OMNIS Software, Inc., and its licensors 1998. All rights reserved.
Portions © Copyright Microsoft Corporation.

OMNIS® is a registered trademark and OMNIS 5™, OMNIS 7™, and OMNIS Studio are trademarks of OMNIS Software, Inc.

Microsoft, MS, MS-DOS, Visual Basic, Windows, Windows 95, Win32, Win32s are registered trademarks, and Windows NT, Visual C++ are trademarks of Microsoft Corporation in the US and other countries.

Apple, the Apple logo, AppleTalk, and Macintosh are registered trademarks and MacOS, Power Macintosh and PowerPC are trademarks of Apple Computer, Inc.

IBM and AIX is a registered trademark and OS/2 is a trademark of International Business Machines Corporation.

UNIX is a registered trademark in the US and other countries exclusively licensed by X/Open Company Ltd.

Sun, Sun Microsystems, the Sun Logo, Solaris, Java, and Catalyst are trademarks or registered trademarks of Sun Microsystems Inc.

HP-UX is a trademark of Hewlett Packard.

OSF/Motif is a trademark of the Open Software Foundation.

Acrobat is a trademark of Adobe Systems, Inc.

ORACLE is a registered trademark and SQL*NET is a trademark of Oracle Corporation.

SYBASE, Net-Library, Open Client, DB-Library and CT-Library are registered trademarks of Sybase Inc.

INFORMIX is a registered trademark of Informix Software, Inc.

EDA/SQL is a registered trademark of Information Builders, Inc.

CodeWarrior is a trade mark of Metrowerks, Inc.

Other products mentioned are trademarks or registered trademarks of their corporations.

Table of Contents

ODBC DRIVER FOR OMNIS DATA FILES	5
OVERVIEW.....	5
<i>Conformance and Requirements.....</i>	<i>5</i>
<i>Configuring a Data Source.....</i>	<i>6</i>
<i>Transactions and Cursors.....</i>	<i>6</i>
<i>Statements and Errors</i>	<i>6</i>
<i>Table Access</i>	<i>6</i>
DATATYPES	7
<i>Limitations.....</i>	<i>8</i>
ODBC API CALLS	8
<i>SQLAllocHandle.....</i>	<i>8</i>
<i>SQLBindCol.....</i>	<i>9</i>
<i>SQLBindParameter</i>	<i>9</i>
<i>SQLCancel.....</i>	<i>10</i>
<i>SQLCloseCursor.....</i>	<i>10</i>
<i>SQLColAttribute</i>	<i>10</i>
<i>SQLColumns.....</i>	<i>10</i>
<i>SQLConnect.....</i>	<i>11</i>
<i>SQLCopyDesc.....</i>	<i>11</i>
<i>SQLDataSources</i>	<i>11</i>
<i>SQLDescribeCol.....</i>	<i>11</i>
<i>SQLDisconnect.....</i>	<i>11</i>
<i>SQLDriver</i>	<i>11</i>
<i>SQLDriverConnect</i>	<i>12</i>
<i>SQLEndTran.....</i>	<i>12</i>
<i>SQLExecDirect.....</i>	<i>12</i>
<i>SQLExecute</i>	<i>12</i>
<i>SQLFetch.....</i>	<i>12</i>
<i>SQLFetchScroll</i>	<i>13</i>
<i>SQLFreeHandle.....</i>	<i>13</i>
<i>SQLFreeStmt</i>	<i>13</i>
<i>SQLGetConnectAttr.....</i>	<i>13</i>
<i>SQLGetCursorName.....</i>	<i>13</i>
<i>SQLGetData</i>	<i>13</i>
<i>SQLGetDescField.....</i>	<i>14</i>
<i>SQLGetDescRec</i>	<i>14</i>
<i>SQLGetDiagField.....</i>	<i>14</i>
<i>SQLGetDiagRec</i>	<i>15</i>
<i>SQLGetEnvAttr.....</i>	<i>15</i>

<i>SQLGetFunctions</i>	15
<i>SQLGetInfo</i>	15
<i>SQLGetStatementAttr</i>	15
<i>SQLGetTypeInfo</i>	15
<i>SQLNativeSql</i>	16
<i>SQLNumParams</i>	16
<i>SQLNumResultCols</i>	16
<i>SQLParamData</i>	16
<i>SQLPrepare</i>	16
<i>SQLPutData</i>	17
<i>SQLRowCount</i>	17
<i>SQLSetConnectAttr</i>	17
<i>SQLSetCursorName</i>	17
<i>SQLSetDescField</i>	17
<i>SQLSetDescRec</i>	17
<i>SQLSetEnvAttr</i>	18
<i>SQLSetStatementAttr</i>	18
<i>SQLSpecialColumns</i>	18
<i>SQLStatistics</i>	19
<i>SQLTables</i>	19
<i>Macros</i>	19
BACKWARDS COMPATIBILITY	20
<i>SQLExtendedFetch</i>	20
<i>Date, Time and TimeStamp Datatypes</i>	20
SQL GRAMMAR	20

ODBC Driver for OMNIS Data Files

Overview

The ODBC driver implements functionality specified by version 3 of the Microsoft ODBC API to allow read-only access to either an OMNIS 7 or Studio datafile. This document describes the minimum functionality provided by the driver to allow its conformance to the core ODBC V3 API. For specific details of this API refer to the ODBC Programmer's Reference¹.

Conformance and Requirements

The ODBC driver supports applications written to conform to both the ODBC V2.0 and V3.0 APIs, but only provides functionality at the V3.0 core level. The Driver Manager maps V2.0 function calls to their V3.0 equivalents for backwards compatibility. The driver requires an ODBC Driver Manager V3.0 such as those provided by Microsoft and Intersolve.

The driver is available for Windows 32 bit and Macintosh PowerPC platforms. All operations are performed synchronously. Under Windows 32 bit platform, the driver can be used in a multi-threaded environment but only allows a single thread to be running at any one time.

The driver requires the ODBC 3.0 Driver Manager since it conforms to the version v3.0 ODBC API. Under Windows, this is supplied as the data access component of Microsoft Office 97.

¹ ODBC 3.0 Programmer's Reference.
Microsoft website.

Microsoft Corporation, downloaded from the

Configuring a Data Source

After installation you can configure each OMNIS ODBC Data Source from the standard ODBC control panel. The driver supports file, user and system data sources.

When you add or modify an OMNIS data source, the driver displays a dialog which lets you enter the following:

1. The name of the data source.
2. A description of the data source.
3. The path name of the OMNIS data file to be used by the data source. This can be left empty meaning that the driver prompts for a data file as part of the connection process.

Transactions and Cursors

Transactions are not supported in this read-only driver.

The driver supports block, forward-only cursors. A cursor is associated with each SELECT statement and the name of the cursor is either defined by the application, see **SQLSetCursorName**, or uniquely generated by the driver. A cursor is open as long as the statement execution is successful and a result set is generated. Cursors are closed when using **SQLCloseCursor** or **SQLFreeStmt**. Cursors must be closed before a SELECT statement is re-executed.

Statements and Errors

Since this is a read-only driver the SQL grammar supports only the SELECT statement to generate result sets.

Relevant error and diagnostic information is available, see **SQLGetDiagRec** and **SQLGetDiagField**.

Table Access

Access to a table slot in an OMNIS data file using the ODBC driver is configurable using the OMNIS SQL Browser in both OMNIS 7 and Studio. The default in OMNIS is that a table slot is not accessible from the driver.

DataTypes

The following is a list of OMNIS data types and their mappings to the ODBC types supported by the driver.

OMNIS DataType	OMNIS SubType	ODBC SQLType	Default ODBC C Data
Character	-	SQL_VARCHAR	SQL_C_CHAR
		SQL_LONGVARCHAR (length > 255 characters)	SQL_C_CHAR
National	-	SQL_VARCHAR	SQL_C_CHAR
		SQL_LONGVARCHAR (length > 255 characters)	SQL_C_CHAR
Number	Short Integer	SQL_TINYINT (unsigned)	SQL_C_UTINYINT
	Long Integer	SQL_INTEGER (signed)	SQL_C_SLONG
	Short 0 dp	SQL_SMALLINT (signed)	SQL_C_SSHORT
	Short n dp	SQL_DOUBLE	SQL_C_DOUBLE
	Number n dp	SQL_DOUBLE	SQL_C_DOUBLE
	Floating dp	SQL_DOUBLE	SQL_C_DOUBLE
Boolean	-	SQL_BIT	SQL_C_BIT
Date Time	Date time	SQL_TYPE_TIMESTAMP	SQL_C_TYPE_TIMESTAMP
	Short date	SQL_TYPE_DATE	SQL_C_TYPE_DATE
	Short time	SQL_TYPE_TIME	SQL_C_TYPE_TIME
Sequence	-	SQL_INTEGER (unsigned)	SQL_C_ULONG

Note that Binary, Picture, List, Item Reference, Row, and Object columns are not accessible using the driver.

The driver allows result set columns to be converted from the supported ODBC SQL types listed above to all supported ODBC C types where that conversion is a valid one, see **SQLBindCol**. Parameters can also be converted from the supported ODBC C types to the ODBC SQL types, see **SQLBindParameter**.

Limitations

The driver does not allow the use of the following concise types for data conversion.

SQL_C_SBIGINT, SQL_C_UBIGINT

SQL_C_BOOKMARK, SQL_C_VARBOOKMARK

SQL_C_NUMERIC

Any of the SQL_C_INTERVAL types.

The OMNIS Short time type does not retain seconds information passed from SQL_TYPE_TIME values. If seconds are significant, the application should map the data to a SQL_TYPE_TIMESTAMP value and the driver converts to that type.

ODBC API Calls

The ODBC driver supports the following V3.0 core level ODBC API calls; they are listed in alphabetical order. Further details can be found in the Microsoft ODBC 3.0 Programmer's Reference. The Driver Manager maps calls made in a V2.0 application to these functions.

SQLAllocHandle

Allocates environment, connection, statement, and explicit descriptor handles. This call can be used to allocate handles to manage access to a data source at differing levels. An environment handle references a global context used to control multiple connections to a driver and its data sources. A connection handle manages each connection within an environment and its associated statements and descriptors. A statement handle is used to manage a SQL statement generated on a connection. An explicit descriptor handle is used to set and retrieve information about statement parameters and result set columns within a connection.

There can be multiple allocations of each type and the driver allows the same handles to be used by different threads. The number of connections and active statements is limited only by system resources (SQLGetInfo(SQL_MAX_CONCURRENT_ACTIVITIES) and SQLGetInfo(SQL_MAX_DRIVER_CONNECTIONS) both return 0).

SQLBindCol

Allows binding of data buffers allocated by the application to result set columns. A bind offset value can be used by changing the `SQL_ATTR_ROW_BIND_OFFSET_PTR` statement attribute. This allows the address of the buffer to be changed without re-issuing this call. Column and row array binding is allowed. Conversions from the source SQL type to all supported ODBC C types are allowed, see **DataTypes**.

Limitations

This call uses the column number to identify the column to bind. These column numbers start at 1 since column 0 is reserved for bookmark columns and the driver does not support bookmark information.

SQLBindParameter

Binds a single buffer allocated by the application to a parameter marker in a SQL statement. A rebinding of parameters to a buffer can occur by specifying a bind offset using the `SQL_ATTR_PARAM_BIND_OFFSET_PTR` statement attribute. Values are read from the buffers when the statement is executed.

Alternatively, this call can be used in conjunction with **SQLParamData** and **SQLPutData** to send data at execution parameters. These can be of any type or size, but this call is useful when large amounts of data, usually character or binary are to be passed in chunks. The application passes a parameter identifier, e.g. a number instead of a buffer pointer, with **SQLBindParameter**. After execution is initiated with **SQLExecute** you can query the driver using **SQLParamData** to determine which parameter data should be sent using **SQLPutData**.

Valid conversions are allowed from ODBC C types to the SQL types relevant to an OMNIS datafile.

Limitations

The type of the parameter to bind as defined by the `InputOutputType` argument of this call can only be one of `SQL_PARAM_INPUT` or `SQL_PARAM_INPUT_OUTPUT` (treated as `INPUT`), since the driver does not support output parameters.

Arrays of parameters are not supported and `SQLGetInfo(SQL_PARAM_ARRAY_SELECTS)` returns `SQL_PAS_NO_SELECT`.

SQLCancel

Cancels the current execution of a statement. The context in which this call would be used is where SQLExecute has been called for a statement and data-at-execution parameters are required. This call can be made for any valid statement handle.

SQLCloseCursor

Closes an open cursor associated with a SELECT statement and discards all pending results for that cursor. This call must be made before re-executing a SELECT statement.

SQLColAttribute

Returns descriptor (meta-data) information about a result set and its columns.

Every column in a result set has an associated descriptor record. This record consists of fields which contain information such as the actual length of a column, whether the column allows null values, etc. The application determines the information it requires by passing the column number and field identifier for that attribute. This call is only valid where a statement returning a result set has been prepared or executed.

Limitations

Column references can only start at one since the driver does not support bookmark information. Only the core descriptor fields are supported.

SQLColumns

Retrieves column meta-data information about a single column or all columns in a single table or all tables in a datafile. This call returns a result set of description rows containing data such as the table name, column name, column type, etc, which can be processed like any other. The application can use pattern matching arguments for this calls *TableName* and *ColumnName* parameters.

Limitations

The driver does not support the use of catalog or schema names in this call and they must be passed as NULL pointers with lengths of zero. Result set columns which are not supported return NULL or empty.

Binary, Picture, List, Item Reference, Row and Object columns are not returned by the driver.

SQLConnect

Connects to the driver and data source. The application provides the data source name in this calls *ServerName* parameter. The application provides a connection handle previously generated using `SQLAllocHandle`.

SQLCopyDesc

If the application wishes to duplicate parameter and column descriptor settings, it calls this function. The application provides a source and destination descriptor handle.

SQLDataSources

Returns information about a data source currently supported by the Driver Manager. This is repeatedly called to return details of each data source.

SQLDescribeCol

Is similar to `SQLColAttribute`, but returns multiple descriptor information about a column in a result set. For the specified column number this call returns the column name, column type, column size, number of decimal digits, and whether or not the column can be null. This information is returned directly to buffers the application provides when making this call. If a column type is non-decimal the number of digits returned is zero. This information is also obtained by calling `SQLColAttribute` for each related descriptor field.

Limitations

Column references can only start at one since the driver does not support bookmark information.

SQLDisconnect

Closes the connection opened on a connection handle. If statements remain allocated within the connection they are freed.

SQLDriver

Provides the calling application with a description of each available driver and it's attributes. This information is passed back in buffers provided with the call.

SQLDriverConnect

Used as an alternative to SQLConnect where the application provides a connection string to determine which datafile to connect to. Where the call is successful and the connection string is incomplete the complete connection string is returned in a string buffer provided by the application. The driver supports the standard keyword, DSN, for the Data Source Name. The driver defines the DataFilePath keyword to allow the path name to be specified in the connection string. A dialog box is displayed to prompt for this information if either of these are omitted. Other keywords such as UID or PWD are not supported.

SQLEndTran

Simulates a commit of all transactions within an environment or connection. This call is supported for the SQL_COMMIT option only, but is redundant since the driver does not support transactions. Issuing this call does not close a cursor or clear result sets.

SQLExecDirect

Submits a SQL statement which is immediately prepared and executed. The driver makes any changes to the statement to match its grammar and substitutes values where parameter markers are used. The statement is always parsed when this call is made, so it is best used for statements that are not re-executed. For a discussion of permissible SQL statements see **SQL GRAMMAR**. Valid data conversions can occur for input parameters.

SQLExecute

Executes a previously prepared statement. Parameter values are substituted for any parameters in the statement. Statements can be re-executed with new parameter values once previous result sets have been processed or discarded. Valid data conversions can occur for input parameters.

SQLFetch

Retrieves one or more rows from the current result set. Data is retrieved and converted into any data buffers bound by the application using **SQLBindCol** and the associated length/indicator buffer is updated. The status of each row is returned in the row status array and the number of rows fetched is placed in the rows fetched buffer. Multiple rows are retrieved if the SQL_ATTR_ROW_ARRAY_SIZE statement attribute is greater than one.

SQLFetchScroll

Alternative call to **SQLFetch**, functionality is identical.

Limitations

The driver only supports the *FetchOrientation* argument of `SQL_FETCH_NEXT` and ignores the *FetchOffset* value.

SQLFreeHandle

Frees environment, connection, statement, and descriptor handles.

SQLFreeStmt

Performs all or one of the following; end a statement's execution, close an open cursor and clear bindings for input parameters or result columns associated with a statement.

ODBC V3.0 provides the **SQLCloseCursor** call which should be used to close a cursor in a V3 application.

SQLGetConnectAttr

Provides current connection attribute values to the application. See **SQLSetConnectAttr** for attribute limitations.

Limitations

The value returned by querying the read-only attribute `SQL_ATTR_AUTO_IPD` is always `SQL_FALSE`, since the driver does not support the automatic population of parameter descriptors for a prepared statement.

SQLGetCursorName

Returns the name of the current cursor associated with a statement. This is set either by the application using **SQLSetCursorName** or is generated by the driver. A driver generated name begins with `SQL_CUR` and is up to `SQL_MAX_ID_LENGTH` characters. `SQL_MAX_ID_LENGTH` is defined by the driver as 128.

SQLGetData

Retrieves data from a result set column. It is mainly used to retrieve long data values, such as variable length binary and text, in chunks, but can be used for any datatype of any size. The call is repeatedly used to process all the data. It is generally used when the result

column is not bound to an application buffer using **SQLBindCol** but can be used for bound columns. Any valid data conversion occurs from the source to the *TargetType* specified by the application. Column data can be retrieved from a row in any order.

Limitations

Columns references can only start at one since the driver does not support bookmark information. The driver does not support the use of **SQLGetData** to retrieve data from a rowset containing multiple rows, i.e. `SQL_ATTR_ROW_ARRAY_SIZE` is greater than one. The `SQLGetInfo(SQL_GETDATA_EXTENSIONS)` returns the following bitmasks; `SQL_GD_ANY_COLUMN`, `SQL_GD_ANY_ORDER`, `SQL_GD_BOUND`.

SQLGetDescField

Gets the current value of a single field in an input parameter or rowset column descriptor. This is similar to **SQLGetDescRec** but every descriptor field can be queried including fields in the descriptor header.

Limitations

Bookmarks descriptors are unsupported.

SQLGetDescRec

Returns the current values of multiple fields in the descriptor record for an input parameter or rowset column. These values consist of the name, type and storage settings for a particular parameter or column. This information describes data buffers used to pass information between the driver and datafile. The application provides a handle to the descriptor and the field values are returned in buffers passed with the call.

Limitations

This call can not be used to retrieve information about the bookmark column zero, since bookmarks are unsupported.

SQLGetDiagField

Retrieves individual error, warning and status fields from a diagnostic record associated with an environment, connection, statement or descriptor handle. This is used to describe any error, warning or informational states resulting from a command. All relevant fields are supported.

SQLGetDiagRec

Provides a set of information from a diagnostic record associated with an environment, connection, statement or descriptor handle. This includes the error code and text associated with the last command performed on that handle. The application provides buffers at the time of calling in which to report these diagnostics.

SQLGetEnvAttr

Queries any of the current values of the supported environment attributes.

SQLGetFunctions

Returns information about which ODBC functions the driver supports. The application can either query a single function or all functions. This call returns `SQL_TRUE` or `SQL_FALSE` if the function is supported or not. A V3.0 application uses the `SQL_API_ODBC3_ALL_FUNCTIONS` option to gain information about all V3 and earlier functions and a V2.0 application uses the `SQL_API_ALL_FUNCTIONS` for all V2 and earlier functions. The application provides an array in which to store the supported status of more than one function. All functions in the V3.0 core conformance level, as specified in this document, return `SQL_TRUE`.

SQLGetInfo

Returns information about the driver and data source. The application specifies the type of information it requires, e.g. `SQL_MAX_COLUMNS_IN_SELECT` to retrieve the maximum number of columns allowed in a SQL `SELECT` statement. All V3.0 types are supported, however, not all types are applicable to the driver and in that case the driver returns a default value. For example, a query on the maximum catalog name length returns zero (`SQL_MAX_CATALOG_NAME_LEN`), since an OMNIS datafile does not contain catalog information. Some of the V2.0 and V1.0 information types are deprecated in V3.0 and should not be used in a V3.0 application.

SQLGetStatementAttr

Retrieves the current settings of supported statement level attributes.

SQLGetTypeInfo

Returns the data types the driver supports. The application can specify a SQL data type to return specific information or `SQL_ALL_TYPES` for information on all supported types. The data is returned as a result set and is processed accordingly. This call provides

information such as the data source type, the associated SQL type, the column size, etc. See **Data Types** for a list of supported types.

Limitations

The driver does not support the result set column `CREATE_PARAMS` which describes the column syntax used when creating a table. This column always returns empty since the driver is read-only.

SQLNativeSql

Returns the SQL statement which would be executed after any substitutions have occurred, e.g. escape sequences converted to the driver's native SQL.

SQLNumParams

Returns the number of parameters in a SQL statement.

SQLNumResultCols

Where a SQL statement generates a result set, this call returns the number of columns in that set. This can only be used if a statement has been prepared or executed. An empty result set still provides column number information.

SQLParamData

Determines the data an application should provide at execution.

If an application requires that data be sent at execution, a previous call to **SQLBindParameter** sends a parameter marker identifier to the driver. The application also calls **SQL_LEN_DATA_AT_EXEC** to tell the driver that a value is to be provided at execution time. An application can check the status of the **SQLExecute** call. If this returns `SQL_NEED_DATA` the application has specified that a parameter value be provided at execution. The application calls **SQLParamData** to retrieve the identifier and using this reference sends the correct data using **SQLPutData**. An application tests the status of **SQLParamData** and if this returns `SQL_NEED_DATA` more data is expected.

SQLPrepare

Prepares a SQL statement for execution. The statement can be repeatedly executed using **SQLExecute**. The statement is parsed and any syntax errors are generated after this call.

SQLPutData

Used in conjunction with **SQLParamData** when an application is to provide parameter data at execution. This allows chunks of character or binary data to be sent to build up larger values.

SQLRowCount

Determines the number of rows in the result generated by the last **SELECT** command executed. The driver does not report this information and returns zero.

SQLSetConnectAttr

Sets a connection attribute in order to configure a connection.

Limitations

The attribute value of `SQL_ATTR_ACCESS_MODE` defaults to `SQL_MODE_READ_ONLY` since the driver only supports read access. All non-core attributes are unsupported.

SQLSetCursorName

Defines the cursor name to be used for the current statement. This can be no longer than `SQL_MAX_ID_LENGTH` characters. Values in quotes are treated as case sensitive and can include characters which are invalid to the SQL syntax.

SQLSetDescField

Sets the current value of a single field in an input parameter or rowset column descriptor. Other functions, e.g. **SQLBindParameter** affect these fields when called. Some fields are read-only.

Limitations

Bookmarks descriptors are unsupported. Non-core descriptor fields are unsupported.

SQLSetDescRec

Sets multiple fields in a parameter or column's descriptor record. This information consists of the name, type, and storage settings for a particular parameter or column. This affects the way in which data is treated when passed to and from the datafile. A particular descriptor is identified by its handle, which is passed with the call.

Limitations

Bookmarks descriptors are unsupported.

SQLSetEnvAttr

Sets any of the supported environment attributes. Attributes are only supported at the core level.

SQLSetStatementAttr

Returns the current setting of any supported statement attribute.

Limitations

All non-core attributes are unsupported. Setting the following core level attributes has no effect as they are not relevant to the driver.

SQL_ATTR_PARAM_OPERATION_PTR

Allows parameter values to be disregarded at execution.

SQL_ATTR_PARAM_STATUS_PTR

Status of each parameter set.

The SQL_ATTR_PARAM_SET_SIZE is always one.

SQLSpecialColumns

Returns a row identifier. This consists of a rowset containing the columns in a table which uniquely identify a row, i.e., the columns in a unique index. This call can only be made with an *IdentifierType* of SQL_BEST_ROWID. Other options can be set by the application to control whether identifiers allowing null values are returned and the scope of the row identifier.

Limitations

This call does not support the use of the SQL_ROWVER option which specifies that columns which are automatically updated be returned. You can pass any of the scope options for the rowid returned. However, the driver does not support transactions and therefore the value of SCOPE in the result set is always one of SQL_SCOPE_CURROW, i.e., rowid values may have changed since the last SELECT. The driver does not support the use of catalog and schema names.

SQLStatistics

Retrieves information about a single table and its indexes. The application can specify whether information about unique indexes or all indexes is returned. This call generates a result set with rows corresponding to each index with information such as index name, uniqueness, etc. When the `SQL_ATTR_METADATA_ID` statement attribute is set to `SQL_FALSE` this calls *TableName* parameter is treated as an ordinary argument and it's case is significant. When set to `SQL_TRUE` this argument is treated as an identifier and it's case is not significant.

Limitations

This call does not support the use of catalog and schema names and these should be passed as NULL pointers with a length of zero. The driver does not return cardinality or page information for a table and these are always NULL.

SQLTables

Lists information about the tables in a datafile. The application can specify whether information about a single table or all tables are returned. This call can also be used to retrieve a result set related to specific table types, if it's *TableType* parameter is a list of types, e.g. "TABLE,VIEW", etc, or all types when set to `SQL_ALL_TABLE_TYPES`. The values specified for this calls *TableName* parameter can include pattern matching.

Limitations

This call does not support the use of catalog and schema names and these should be passed as NULL pointers with a length of zero. This call only supports the use of TABLE as a type since this is all that applies to an OMNIS datafile. Result set columns which are not supported return NULL or empty., e.g. catalog and schema names.

Macros

SQL_LEN_DATA_AT_EXEC

Used after **SQLBindParameter** is called to denote that a parameter value will be provided at execution. The length of the long parameter data supplied with this call is not required by the driver and `SQLGetInfo(SQL_NEED_LONG_DATA_LEN)` returns "N".

SQL_FUNC_EXISTS

A call to **SQLGetFunctions** with an option of `SQL_API_ODBC3_ALL_FUNCTIONS` provides an array in its *SupportedPtr* argument which contains all supported functions. This macro is called after **SQLGetFunctions** and uses the *SupportedPtr* array with a function identifier to check whether a particular function exists.

Backwards Compatibility

The following functionality is provided to enable a V2.0 application to use the driver. An application can set the `SQL_ATTR_ODBC_VERSION` attribute to `SQL_OV_ODBC2` to enable V2.0 behavior in a V3.0 driver, although this is not explicitly required in a V2.0 application.

SQLExtendedFetch

This call is used by a V2.0 application to return a rowset of data from the data file. It functions in a similar way to **SQLFetchScroll** but **SQLExtendedFetch** places the rows fetched and the row status in buffers provided with the call. In addition, this call uses the value of the `SQL_ROWSET_SIZE` statement attribute to control the number of rows returned and bind offsets are not supported.

Limitations

This call does not support the retrieval of bookmark columns.

Date, Time and TimeStamp Datatypes

When exhibiting V2.0 behavior, the ODBC driver supports the SQL date types of `SQL_DATE`, `SQL_TIME`, and `SQL_TIMESTAMP` with concise types of `SQL_C_DATE`, `SQL_C_TIME`, and `SQL_C_TIMESTAMP`.

SQL Grammar

The ODBC driver uses the existing SQL Grammar for `SELECT` statements as used with the OMNIS SQL DAM². This includes the scalar functions that are part of OMNIS SQL. The grammar is extended to conform to the minimum ODBC grammar for `SELECT` statements, as defined in the Microsoft ODBC 3.0 Programmer's Reference. The driver does not support any other forms of statement and if submitted these generate syntax errors. The minimum grammar is a subset of the ANSI SQL-92 standard.

² OMNIS Studio Cache manual (v1), or OMNIS Programming manual (v2) for OMNIS Studio, or the Developer's Guide for OMNIS 7 (OMNIS Software Inc.)

The general form of a SELECT statement is as follows,

```
SELECT [ALL|DISTINCT] {value-expression-list | *}  
      FROM table-reference-list  
      [WHERE search-condition]  
      [GROUP BY column-reference-comma-list [HAVING search-condition]]  
      [ORDER BY order-column-comma-list]
```

The ODBC conformance requirements have added the following features to the OMNIS SQL grammar.

1. SQL-92 features.
 - Expression numbers in an ORDER BY clause.
 - Dynamic parameter makers obey the rules for parameter markers.
 - The use of two quote characters to insert a single quote in a string literal.
2. ODBC features.
 - Escape sequences for date, time and timestamp datatypes.
 - Numeric literals conform to the ODBC specification.

Other components not required by the minimum grammar retain the same behavior and restrictions as in the OMNIS SQL grammar and do not comply to the ODBC/SQL-92 standards. For example, a HAVING clause cannot be used without an associated GROUP BY clause. To determine the support of a particular feature use the SQLGetInfo call.