Microsoft Access Reader/Writer

Format Note: This format is not supported by FME Base Edition.

Overview

The Microsoft® Access reader and writer provide FME with access to attribute data held in live MS Access database tables. This data may not necessarily have a spatial component to it. FME provides read and write access to live MS Access databases via Microsoft's ActiveX Data Objects (ADO).

Note: *Only the standard SQL wildcard characters (% and _) are supported for SQL LIKE queries. Microsoft Access wildcard characters (*, ?, and #) are not supported.*

Note: *See the @SQL function in the FME Functions and Factories manual. This function allows arbitrary Structured Query Language (SQL) statements to be executed against any database.*

MS Access Database Quick Facts

About Quick Facts Tables

Format Type Identifier	MDB_ADO
Reader/Writer	Both
Licensing Level	Professional
Dependencies	▪ File versions prior to 2007: None, but the format is available only on Windows. ▪ File versions 2007 or newer: install a corresponding or newer version of Microsoft Office, or the free download of Microsoft Access Database Engine 2010 Redistributable.
Dataset Type	Database
Feature Type	Table name
Typical File Extensions	Not applicable
Automated Translation Support	Yes
User-Defined Attributes	Yes
Coordinate System Support	No
Generic Color Support	No
Spatial Index	Never
Schema Required	Yes

Transaction Support	Yes
Encoding Support	Yes
Geometry Type	db_none

Geometry Support				
Geometry	Supported?		Geometry	Supported?
aggregate	no		point	no
circles	no		polygon	no
circular arc	no		raster	no
donut polygon	no		solid	no
elliptical arc	no		surface	no
ellipses	no		text	no
line	no		z values	n/a
none	yes			

Reader Overview

FME considers a database data set to be a collection of relational tables. The tables must be defined in the mapping file before they can be read. Arbitrary where clauses and joins are fully supported.

Reader Directives

The suffixes listed are prefixed by the current <ReaderKeyword> in a mapping file. By default, the <ReaderKeyword> for the MS Access reader is MDB_ADO.

DATASET

Required/Optional: *Required*

This is the file name of the Microsoft Access Database.

Example:

```
MDB_ADO_DATASET c:/data/citySource.mdb
```

Workbench Parameter: *Source Microsoft Access Database File(s)*

PROVIDER_TYPE

Required/Optional: *Optional*

The type of database provider being used. This directive is internal to FME and should always be set to MDB_ADO. For example,

```
MDB_ADO_PROVIDER_TYPE MDB_ADO
```

PASSWORD

Required/Optional: *Optional*

The password used to access the database. It can be omitted for Access databases without password protection.

Please note that databases associated with a Microsoft Access workgroup are not supported.

Example:

```
MDB_ADO_PASSWORD moneypenny
```

Workbench Parameter: *Password*

DEF

Required/Optional: *Required*

The syntax of the definition is:

```
MDB_ADO_DEF <tableName> \
       [mdb_where_clause<whereClause>] \
[<fieldName><fieldType>] +
```

or

```
MDB_ADO_DEF <queryName> \
       [mdb_sql_statement <sqlQuery>]  \
```

The `<tableName>` must match the name of an existing MS Access table in the database. This will be used as the feature type of all the features read from the table. The exception to this rule is when using the mdb_sql_statement directive. In this case, the `DEF` name may be any valid alphabetic identifier; it does not have to be an existing table name – rather, it is an identifier for the custom SQL query. The feature type of all the features returned from the SQL query are given the query name.

The `<fieldType>` of each field must be given, but it is not verified against the database definition for the field. In effect, it is ignored.

The definition allows specification of separate search parameters for each table. If any of the per table configuration parameters are given, they will override, for that table, whatever global values have been specified by the reader directives such as the `WHERE_CLAUSE`. If any of these parameters is not specified, the global values will be used.

The following table summarizes the definition line configuration parameters:

Parameter	**Contents**
mdb_where_clause	This specifies the SQL WHERE clause applied to the attributes of the layer’s features to limit the set of features returned. If this is not specified, then all the rows are returned. This directive will be ignored if

Parameter	Contents
	the mdb_sql_statement is present.
mdb_sql_statement	This specifies an SQL SELECT query to be used as the source for the results. If this is specified, the MS Access reader will execute the query, and use the resulting rows as the features instead of reading from the table <queryName>. All returned features will have a feature type of <queryName>, and attributes for all columns selected by the query. The mdb_where_clause is ignored if mdb_sql_statement is supplied. This form allows the results of complex joins to be returned to FME.

If no `<whereClause>` is specified, all rows in the table will be read and returned as individual features. If a `<whereClause>` is specified, only those rows that are selected by the clause will be read. Note that the `<whereClause>` does not include the word WHERE.

The MS Access reader allows one to use the mdb_sql_statement parameter to specify an arbitrary SQL SELECT query on the DEF line. If this is specified, FME will execute the query, and use each row of data returned from the query to define at least one feature. Each of these features will be given the feature type named in the DEF line, and will contain attributes for every column returned by the SELECT. In this case, all DEF line parameters regarding a WHERE clause or spatial querying are ignored, as it is possible to embed this information directly in the text of the <sqlQuery>.

In the following example, the all records whose ID is less than 5 will be read from the supplier table:

```
MDB_ADO_DEF supplier    \
    mdb_where_clause "id < 5" \
    ID  integer  \
    NAME char(100)     \
    CITY char(50)
```

In this example, the results of joining the employee and city tables are returned. All attributes from the two tables will be present on each returned feature. The feature type will be set to complex.

```
MDB_ADO_DEF complex  \
       mdb_sql_statement  \
               "SELECT * FROM EMPLOYEE, CITY WHERE EMPLOYEE.CITY = CITY.NAME"
```

WHERE_CLAUSE

Required/Optional

Optional

This optional specification is used to limit the rows read by the reader from each table. If a given table has no mdb_where_clause or mdb_sql_statement specified in its DEF line, the global <ReaderKeyword>_WHERE_CLAUSE value, if present, will be applied as the WHERE specifier of the query used to generate the results. If a table's DEF line does contain its own mdb_where_clause or mdb_sql_statement, it will override the global WHERE clause.

The syntax for this clause is:

```
MDB_ADO_WHERE_CLAUSE <whereClause>
```

Note: *The <whereClause> does not include the word "WHERE."*

The example below selects only the features whose lengths are more than 2000:

```
MDB_ADO_WHERE_CLAUSE LENGTH > 2000
```

Workbench Parameter

Where Clause

IDs

Required/Optional: *Optional*

This optional specification is used to limit the available and defined database tables that will be read. If no IDs are specified, then all tables are read. The syntax of the IDs directive is:

```
MDB_ADO_IDs <featureType1> \
<featureType2> … \
<featureTypeN>
```

The feature types must match those used in DEF lines.

The example below selects only the HISTORY table for input during a translation:

```
MDB_ADO_IDs HISTORY
```

Workbench Parameter: *Feature Types to Read*

READ_CACHE_SIZE

Required/Optional: *Optional*

This directive controls how the reader retrieves rows from the database. This must be a numeric value which must be greater than 0.

The READ_CACHE_SIZE is used to determine the number of rows that are retrieved at one time into local memory from the data source. For example, if the READ_CACHE_SIZE is set to 10, after the reader is opened, the reader will read 10 rows into local memory. As features are processed by the FME, the reader returns the data from the local memory buffer. As soon as you move past the last row available in local memory, the reader will retrieve the next 10 rows from the data source.

This directive affects the performance of the reader, and will result in significantly degraded performance if incorrectly set. The optimum value of this directive depends primarily on the characteristics of individual records and the transport between the database and the client machine. It is less affected by the quantity of rows that are to be retrieved.

By default, the READ_CACHE_SIZE is set to 10. This value has been determined to be the optimal value for average datasets.

Workbench Parameter: *Number of Records to Fetch At A Time*

RETRIEVE_ALL_SCHEMAS

Required/Optional: *Optional*

This directive is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

When set to "Yes", indicates to the reader to return all the schemas of the tables in the database.

If this directive is missing, it is assumed to be "No".

Range: *YES | NO*

Default: *NO*

RETRIEVE_ALL_TABLE_NAMES

Required/Optional: *Optional*

This specification is only applicable when generating a mapping file, generating a workspace or when retrieving schemas in a FME Objects application.

Similar to `RETRIEVE_ALL_SCHEMAS`; this optional directive is used to tell the reader to only retrieve the table names of all the tables in the source database. If RETRIEVE_ALL_SCHEMAS is also set to "Yes", then RETRIEVE_ALL_SCHEMAS will take precedence. If this directive is not specified, it is assumed to be "No".

Range: *YES | NO*

Default: *NO*

EXPOSED_ATTRS

This directive allows the selection of format attributes to be explicitly added to the reader feature type.

This is similar to exposing format attributes on a reader feature type once it has been generated; however, it is even more powerful because it enables schema-driven applications other than Workbench to access and leverage these attributes as if they were explicitly on the schema as user attributes.

The result of picking a list of attributes is a comma-separated list of attribute names and types that will be added to the schema features. Currently all reader feature types will receive the same set of additional schema attributes for a given instance of the reader.

Required/Optional

Optional

Mapping File Syntax

Not applicable. While it is possible for FME Objects applications to invoke this directive, the required format is not documented. This directive is intended for use in our GUI applications (for example, Workbench) only.

Workbench Parameter

Additional Attributes to Expose

Writer Overview

The MS Access writer module stores attribute records into a live relational database. The MS Access writer provides the following capabilities:

- **Transaction Support:** The MS Access writer provides transaction support that eases the data loading process. Occasionally, a data load operation terminates prematurely due to data difficulties. The transaction support provides a mechanism for reloading corrected data without data loss or duplication.
- **Table Creation:** The MS Access writer uses the information within the FME mapping file to automatically create database tables as needed.
- **Writer Mode Specification:** The MS Access writer allows the user to specify what database command should be issued for each feature received. Valid writer modes are INSERT, UPDATE and DELETE. The writer mode can be specified at three unique levels: at the writer level, on the feature type, or on individual features.

Writer Directives

The directives processed by the MS Access Writer are listed below. The suffixes shown are prefixed by the current `<WriterKeyword>` in a mapping file. By default, the `<WriterKeyword>` for the MS Access writer is `MDB_ADO`.

DATASET, PROVIDER_TYPE

The DATASET and PROVIDER_TYPE directives operate in the same manner as they do for the MS Access reader. The remaining writer-specific directives are discussed in the following sections.

Workbench Parameter: *Destination Microsoft Access Database File*

PASSWORD

Required/Optional: *Optional*

The password used to access the database. For existing databases, it may be omitted for Access databases without password protection. If the database does not exist, then the newly created Microsoft Access database will be protected by this password.

Please note that databases associated with a Microsoft Access workgroup are not supported.

```
MDB_ADO_PASSWORD moneypenny
```

Workbench Parameter: *Password*

DEF

Required/Optional: *Required*

Each MS Access table must be defined before it can be written. The general form of a MS Access definition statement is:

```
MDB_ADO_DEF <tableName> \
       [mdb_update_key_columns <keyColumns>]\
       [mdb_drop_table (yes|no)]\
       [mdb_truncate_table (yes|no)]   \
       [mdb_table_writer_mode (inherit_from_writer|insert|

                        update|delete)] \
       [<fieldName><fieldType>[,<indexType>]]+
```

The table definition allows control of the table that will be created. If the fields and types are listed, the types must match those in the database. Fields which can contain NULL values do not need to be listed - these fields will be filled with NULL values.

If the table does not exist, then the field names and types are used to first create the table. In any case, if a `<fieldType>` is given, it may be any field type supported by the target database.

The configuration parameters present on the definition line are described in the following table:

Parameter	**Contents**
tableName	The name of the table to be written. If a table with the specified name exists, it will be overwritten if the mdb_drop_table DEF line parameter is set to YES, or it will be truncated if the mdb_truncate_table DEF line parameter is set to YES. Otherwise the table will be appended. Valid values for table names include any character string devoid of SQL-offensive characters and less than 128 characters in length.
mdb_table_writer_mode	The the default operation mode of the feature type in terms of the types of SQL statements sent to the database.Valid values are `INSERT, UPDATE, DELETE` and `INHERIT_FROM_WRITER`. Note that `INSERT` mode allows

Parameter	**Contents**
	for only INSERT operations where as UPDATE and DELETE can be overwritten at the feature levels. INHERIT_FROM_WRITER simply indicates to take this value from the writer level and not to override it at the feature type level. **Default**:INHERIT_FROM_WRITER
mdb_update_key_columns	This is a comma-separated list of the columns which are matched against the corresponding FME attributes' values to specify which rows are to be updated or deleted when the writer mode is either UPDATE or INSERT. For example: mdb_update_key_columns ID would instruct the writer to ensure that the FME attribute is always matched against the column with the same name. Also, the target table is always the feature type specified in the DEF line. Each column listed with the mdb_update_key_columns directive must be defined with a type on the DEF line, in addition to the columns whose values will be updated by the operation.
mdb_drop_table	This specifies that if the table exists by this name, it should be dropped and replaced with a table specified by this definition. Default: NO
mdb_truncate_table	This specifies that if the table exists by this name, it should be cleared prior to writing. Default: NO
fieldName	The name of the field to be written. Valid values for field name include any character string devoid of SQL-offensive characters and less than 128 characters in length.
fieldType	The type of a column in a table. The valid values for the field type are listed below: yesno memo hyperlink replicationid oleobject integer

Parameter	Contents
	byte long autonumber datetime decimal(width,decimal) single double currency text(width)
indexType	The type of index to create for the column. If the table does not previously exist, then upon table creation, a database index of the specified type is created. The database index contains only the one column. The valid values for the column type are listed below: **indexed:** An index without constraints. **unique:** An index with a unique constraint.

VERSION

Required/Optional: *Required*

This statement tells the MS Access writer what version of database should be created. If the database file already exists, the writer will automatically detect and use the correct version.

Parameter	Contents
<version>	The version of Microsoft Access database file to create. The valid values are listed below: 2000/2002/2003 95/97 2.0 **Default:** 2000/2002/2003

Example:

```
MDB_ADO_VERSION 2000/2002/2003
```

Workbench Parameter: *Version*

START_TRANSACTION

Required/Optional: *Optional*

This statement tells the MS Access writer module when to start actually writing features into the database. The MS Access writer does not write any features until the feature is reached that belongs to <last successful transaction> + 1. Specifying a value of zero causes every feature to be output. Normally, the value specified is zero – a non-zero value is only specified when a data load operation is being resumed after failing partway through.

Parameter	**Contents**
<last successful transaction>	The transaction number of the last successful transaction. When loading data for the first time, set this value to 0. **Default:** 0

Example:

```
MDB_ADO_START_TRANSACTION 0
```

Workbench Parameter: *Start transaction at*

TRANSACTION_INTERVAL

Required/Optional: *Optional*

This statement informs the FME about the number of features to be placed in each transaction before a transaction is committed to the database.

If the MDB_ADO_TRANSACTION_INTERVAL statement is not specified, then a value of 500 is used as the transaction interval.

Parameter	**Contents**
<transaction_interval>	The number of features in a single transaction. **Default:** 500

If the MDB_ADO_TRANSACTION_INTERVAL is set to zero, then feature based transactions are used. As each feature is processed by the writer, they are checked for an attribute called fme_db_transaction. The value of this attribute specifies whether the writer should commit or rollback the current transaction. The value of the attribute can be one of COMMIT_BEFORE, COMMIT_AFTER, ROLLBACK_AFTER or IGNORE. If the fme_db_transaction attribute is not set in any features, then the entire write operation occurs in a single transaction.

Example:

```
MDB_ADO_TRANSACTION_INTERVAL 5000
```

Workbench Parameter: *Transaction interval*

WRITER_MODE

Required/Optional: *Optional*

Note: *For more information, see the chapter Database Writer Mode on page 19.*

This directive informs the MS Access writer which SQL operations will be performed by default by this writer. This operation can be set to `INSERT, UPDATE` or `DELETE`. The default writer level value for this operation can be overwritten at the feature type or table level. The corresponding feature type DEF parameter name is called mdb_table_writer_mode. It has the same valid options as the writer level mode and additionally the value `INHERIT_FROM_WRITER` which causes the writer level mode to be inherited by the feature type as the default for features contained in that table.

The operation can be set specifically for individual feature as well. Note that when the writer mode is set to INSERT this prevents the mode from being interpreted from individual features and all features are inserted unless otherwise marked as UPDATE or DELETE features. These are skipped.

If the MDB_ADO_WRITER_MODE statement is not specified, then a value of INSERT is given.

Parameter	Contents
<writer_mode>	The type of SQL operation that should be performed by the writer. The valid list of values are below: INSERT UPDATE DELETE **Default:** INSERT

Example:

```
MDB_ADO_WRITER_MODE INSERT
```

Workbench Parameter: *Writer Mode*

BEGIN_SQL{n}

Occasionally you must execute some ad-hoc SQL prior to opening a table. For example, it may be necessary to ensure that a view exists prior to attempting to read from it.

Upon opening a connection to read from a database, the reader looks for the directive `<ReaderKeyword>_BEGIN_SQL{n}` (for `n=0,1,2,...`), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the `FME_SQL_DELIMITER` keyword, embedded at the beginning of the SQL block. The single character following this keyword will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
 FME_SQL_DELIMITER ;
DELETE FROM instructors;
DELETE FROM people
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

✲ ***Workbench Parameter***

SQL Statement to Execute Before Translation

END_SQL{n}

Occasionally you must execute some ad-hoc SQL after closing a set of tables. For example, it may be necessary to clean up a temporary view after writing to the database.

Just before closing a connection on a database, the reader looks for the directive `<ReaderKeyword>_END_SQL{n}` (for `n=0,1,2,...`), and executes each such directive's value as an SQL statement on the database connection.

Multiple SQL commands can be delimited by a character specified using the `FME_SQL_DELIMITER` directive, embedded at the beginning of the SQL block. The single character following this directive will be used to split the SQL, which will then be sent to the database for execution. **Note:** Include a space before the character.

For example:

```
 FME_SQL_DELIMITER ;
DELETE FROM instructors;
DELETE FROM people
WHERE LastName='Doe' AND FirstName='John'
```

Multiple delimiters are not allowed and the delimiter character will be stripped before being sent to the database.

Any errors occurring during the execution of these SQL statements will normally terminate the reader with an error. If the specified statement is preceded by a hyphen ("-"), such errors are ignored.

Required/Optional

Optional

✲ ***Workbench Parameter***

SQL Statement to Execute After Translation

INIT_TABLES

Required/Optional: *Optional*

This directive informs the MS Access writer when each table should be initialized. Initialization encompasses the actions of dropping or truncating existing tables, and creating new tables as necessary.

When `INIT_TABLES` is set to `IMMEDIATELY`, the MS Access writer will initialize all tables immediately after parsing the `DEF` lines and opening the database file. In this mode, all tables will be initialized, even if the MS Access writer receives no features for a given table.

When `INIT_TABLES` is set to `FIRSTFEATURE`, the MS Access writer will only initialize a table once the first feature destined for that table is received. In this mode, if the MS Access writer does not receive any features for a given table, the table will never be initialized.

Workbench Parameter: *Initialize Tables*

COMPRESS_AT_END

When this directive is set to “YES”, the MS Access writer will compress the database after all features have been written. This makes use of the existing MDB database option to compact. If COMPRESS_AT_END is not present the database will not be compressed.

Required/Optional

Optional

Values

Yes | No (default)

Mapping File Syntax

```
<WriterKeyword>_COMPRESS_AT_END YES
```

Workbench Parameter

Compress Database When Done

OVERWRITE_FILE

Required/Optional: Required

If set to YES, deletes the existing database before writing.

Workbench Parameter

Overwrite Existing Database

Writer Mode Specification

The MS Access writer allows the user to specify a writer mode, which determines what database command should be issued for each feature received. Valid writer modes are INSERT, UPDATE and DELETE.

Writer Modes

In INSERT mode, the attribute values of each received feature are written as a new database record.

In UPDATE mode, the attribute values of each received feature are used to update existing records in the database. The records which are updated are determined via the mdb_update_key_columns DEF line parameter, or via the fme_where attribute on the feature.

In DELETE mode, existing database records are deleted according to the information specified in the received feature. Records are selected for deletion using the same technique as records are selected for updating in UPDATE mode.

Writer Mode Constraints

In UPDATE and DELETE mode, the fme_where attribute always takes precedence over the mdb_update_key_columns DEF line parameter. If both the fme_where attribute and the mdb_update_key_columns DEF line parameter are not present, then UPDATE or DELETE mode will generate an error.

When the fme_where attribute is present, it is used verbatim as the WHERE clause on the generated UPDATE or DELETE command. For example, if fme_where were set to 'id<5', then all database records with field ID less than 5 will be affected by the command.

When the fme_where attribute is not present, the writer looks for the mdb_update_key_columns DEF line parameter and uses it to determine which records should be affected by the command. Please refer to See "DEF" for more information about the mdb_update_key_columns DEF line parameter.

Writer Mode Selection

The writer mode can be specified at three unique levels. It may be specified on the writer level, on the feature type or on individual features.

At the writer level, the writer mode is specified by the WRITER_MODE directive. This directive can be superseded by the feature type writer mode specification. For more information on this directive, see the chapter Database Writer Mode.

At the feature type level, the writer mode is specified by the mdb_writer_mode DEF line parameter. This parameters supersedes the WRITER_MODE directive. Unless this parameter is set to INSERT, it may be superseded on individual features by the fme_db_operation attribute. Please refer to the DEF line documentation for more information about this parameter.

At the feature level, the writer mode is specified by the fme_db_operation attribute. Unless the parameter at the feature type level is set to INSERT, the writer mode specified by this attribute always supersedes all other values. Accepted values for the fme_db_operation attribute are `INSERT`, `UPDATE` or `DELETE`.

fme_db_transaction

As each feature is processed by the writer, it is checked for an attribute called `fme_db_transaction`.

The value of this attribute specifies whether the writer should commit or rollback the current transaction. The value of the attribute can be one of:

- `COMMIT_BEFORE`
- `COMMIT_AFTER`
- `ROLLBACK_AFTER`
- `IGNORE`

If the `fme_db_transaction` attribute is not set in any features, the entire write operation occurs in a single transaction.

***Note:** To use this capability, the Transaction Interval (for ArcSDE, this is called Features to Write Per Transaction) must be set to `VARIABLE`.*

Feature Representation

Features read from a database consist of a series of attribute values. They have no geometry. The attribute names are as defined in the DEF line if the first form of the DEF line was used. If the second form of the DEF line was used, then the attribute names are as they are returned by the query, and as such may have their original table names as qualifiers. The feature type of each MS Access feature is as defined on its DEF line.

Features written to the database have the destination table as their feature type, and attributes as defined on the DEF line.