Data File Import/Export for Firebird®

USER MANUAL

Table of Contents

1 Introducing dbFile

dbFile is a cross-platform command-line tool for importing and exporting data between Firebird databases and a range of external data file formats. The supported formats are

- delimited data-sensitive
- fixed position data-sensitive
- fixed format text (fixed field position and record size text)

Firebird Versions Supported

dbFile supports all the mainstream, official releases of Firebird distributed directly from the download sections at the Firebird Project main website or the IBPhoenix 'Main Downloads' section. Support for versions built and distributed from other sources cannot be guaranteed. IBPhoenix would be interested to hear of widely-available third-party distributions that exhibit problems with dbFile. It would also be helpful to hear of any problems you encounter when testing this utility with Alpha or Beta versions of forthcoming Firebird releases.

Platforms

Versions are available for Linux, MacOSX and Windows. For other platforms IBPhoenix can build dbFile to order—please inquire.

dbFile is designed to be installed easily and to be run, directly or indirectly, from a command shell. Currently, no graphical user interface version is available. On Linux and MacOSX it can be scripted as a bash script or cron job; on Windows it can be executed in a .bat file or with other scripting tools that can execute shell commands. This documentation assumes the user is familiar with the platform tools and does not attempt to provide instructions for their usage.

Installation

dbFile is distributed as a stand-alone, compiled binary program, without source code. Installation is simply a matter of copying the executable program to your chosen directory and running it from there. An executable installer is available for Windows.

dbFile Licensing

dbFile is proprietary software. While you are running the free trial version, you will see a reminder message each time you call the program:

```
H:\projects\dbFile\app>dbfile -x h:\projects\dbfile\control\myfile1.cfl
This is a trial version of dbFile only, please contact IBPhoenix (http://www.ibp
hoenix.com) to purchase a fully licensed version.
 14 input (or update) records (short: 0, skipped: 0)
        processed from file H:\projects\dbFile\data\dos\countryu.csv

H:\projects\dbFile\app>_
```

You are most welcome to test it and, if you decide to add it to your toolbox, we expect you to purchase the licensed version. If you decide not to use it, you should remove it from your system.

1.1 General Usage Notes

Case Sensitivity

Keywords and switches can be either upper or lower case on any platform.

> Pay attention to any string arguments that are data, such as file paths and user names, and conform to platform rules regarding case-sensitivity. Firebird passwords are always case-sensitive. On most POSIX platforms, system user names and file specifications are unconditionally case-sensitive.

Delimiter Usage

An escape sequence for non-printable characters, such as \t for the tab character that Excel uses as its default field delimiter, is valid and will make the directive easier to read.

If you have free choice about which character to use, the vertical bar, a.k.a. "pipe", | is usually good, since it rarely occurs in user text. It will have to be avoided if text fields are likely to contain the unescaped pipe character. For example, if an incoming data field contains an SQL string expression, the SQL string concatenator, double-pipe || will cause havoc.

Quoted String Fields

If your delimited input data strings have the potential to include literal characters that have syntactic significance in the shell or SQL: environments, such as commas or semi-colons, it is strongly recommended that you quote the strings and include the options quote=<required quote symbol> directive to describe the string-quoting convention to dbFile.

- Excel-format files employ the double-quoting convention on text fields
- For output, strings use the quote option according to requirements. If the excel option is specified, it is recommended to specify the double-quoting

Working Modes

On POSIX, most instructions can be entered into a single line from the command shell 22, which can be quick and simple or complex enough to need a bash script for it. On Windows, an extremely simple import could be achieved from the command line with only a data file but, in general, the command shell in Windows is too weak to make this mode a practicable option.

For complex commands or to store an import or export process that you'll want to use again, dbFile provides an easier and more powerful way to control the operation: use of a control file 13 to package the options and the SQL for the import or export 5.

Character Sets

The dbFile utility application does not provide any special client-side support for the international character set (INTL) features of Firebird, beyond provision of the option charset= syntax for setting the client character set in a control file, along with its command-line switch alternative, -t. Both must take as argument the identifier of a valid character set that is known to the database. It will be your responsibility to ensure that external and database table data are compatible for any transliteration that must occur.

- for imports, dbFile has no way to determine whether text data it is reading from an input

file is well-formed, before passing it across the Firebird client interface for transliteration to the character set defined for the target column

- for exports, the text written will be the same bytes that are stored in the database column unless the byte-by-byte composition of the stored data is modified during retrieval, e.g., by casting

1.2 Date/Time Data

If an import field is to be received as a date or date/time the default format depends on whether you are using the "international" or "U.S." build of dbFile.

- For the international versions, the default format is dd/mm/yyyy
- For the U.S. versions, the default format is mm/dd/yyyy

The same formats apply for exports, according to whether the build is international or U.S.

NOTE :: Imported date fields will also be treated as default format if the separator character is a hyphen instead of a forward slash, viz., dd-mm-yyyy for international, mm-dd-yyyy for U.S. builds.

Non-Default Date Formats

To correctly interpret or output date or date/time strings other than the default, you can include an attribute dateform=N, where N is a number specifying another supported format.

Global Specification of Date Format

Using `options dateform=N` in a control file, or using instead the command-line switch `-f N`, causes all imported date data to be interpreted or all exported date literals to be written according to the date format corresponding to N. If you specify the date format at this level, it is your responsibility to ensure that all of the date fields in incoming data are presented consistently in the format anticipated by N.

If an `options dateform=N` directive is present in the control file, any usage of the switch `-f N` in the command-line call will be ignored.

Field-level Specification of Date Format

If your input or output record specification includes fields of more than one literal format then the `dateform=N` attribute should NOT be specified globally. The attribute should be supplied in the SQL statement, embedded in curly braces after the SQL identifier of the column it is to qualify, for example, `{dateform=N}`. (It might be one of a comma-separated list of attribute qualifiers for the field.)

For more details about embedded attributes, refer to the topic dbFile Embedded Field Descriptors 5 in the chapter entitled The SQL Statement.

Date/Time Formats Supported

N	Template	Other Information

11	yyyy/mm/dd	10-character ISO date format
12	mm/dd/yyyy	10-character U.S. standard date format (default in U. S. builds)
13	yyyy/mm/dd hh:mm:ss	19-character date/time, 24-hour clock
14	mm/dd/yyyy hh:mm:ss	19-character date/time, 24-hour clock
15	dd/MMM/yyyy	11 character date where mmm is an English alpha month abbreviation [JAN/FEB/MAR/APR/MAY/JUN/ JUL/AUG/SEP/OCT/NOV/DEC]
16	dd/mm/yyyy	10-character international date format (default in international builds)
17	dd/mm/yyyy hh:mm:ss	19-character international date format, 24-hour clock

In all the above formats, output will have slash separators unless forced to be hyphens (see below 4)

The remaining templates are external formats that can be applied only to fields in fixed-length records

19	yyyymmdd.hhmmss	double timestamp form
20	yyyymmdd	32-bit integer date form
21	yyyyddd	32-bit integer Julian date form—ddd is day in year
22	yyyymmddhhmmss	64-bit integer form

Forcing Hyphen Separators in Output

By default, all exported date and date/time data that use month and day separators will use slashes. You can force hyphens to be used in place of slashes, as follows:

- in a control file, include the options hyphenateDate directive
- in a single command-line where the operation is not invoking a control file, use the -h switch

NOTE :: It is not possible to generate a record format that has some date formats using slashes and others using hyphens.

1.3 Further Information

Known Issues

There are no known issues at this time.

An updated FAQ page will be posted regularly on the IBPhoenix web site.

Reporting Bugs and Requesting New Features

Please report problems, bugs or issues to the dbFile support list hosted on www.ibphoenix.com.

To subscribe to the list, go to the lists page at the IBPhoenix website and look for dbFile.

2 The SQL Statement

For exports, an SQL SELECTstatement specifies the data you want to extract from the database and pass to the file. For imports, you can use an INSERT, UPDATE, UPDATE OR INSERT statement to specify the write to the database, or you can call a parameterised executable procedure with EXECUTE PROCEDURE in order to perform the desired DML completely at the server side.

For an export, your SELECT can be a call to a selectable procedure that returns the desired data to dbFile. It must be a stored procedure that has been written with a SUSPEND command designed to return one set, being the set that is specified for the output file.

Do not try to call an executable procedure using SELECT.

Statements for Input

For input statements, the standard SQL syntax for specifying unnamed replaceable parameters with a comma-separated list of questionmarks is used. The dbFile parser treats the set of parameters as a group of variables that, by default, are in the same order in the input file. For example, the following statement might be used to load data from a three-field input record in a delimited file into a table called STATES with columns COUNTRY_CD, STATE_CD and STATE_NAME:

```
insert into states (
  country_cd,
  state_cd,
  state_name
  )
values (?,?,?)
```

NOTE that statement terminators—such as ISQL's default semicolon—and the ISQL SET TERM statement are not valid in the dbFile environment.

dbFile Embedded Field Descriptors

dbFile implements some additional syntactic elements that are enclosed in curly braces and embedded into the SQL statement, following the identifier of the column to which the descriptor applies. Multiple attributes for a field are comma-separated. The following example is an element describing some attributes of one incoming field:

```
{position=10, size=2}
```

Such elements, which are required for processing fixed format record input and optional for delimited record input, are stripped out before the DSQL request is submitted to the Firebird engine. The following simple example illustrates the usage of a qualifier element for the same import specified by the previous example:

```
insert into states (
  country_cd {position=10, size=2},
  state_cd {position=8, size=2},
  state_name {position=71, size=35}
                    )
values (?,?,?)
```

> NOTE
> Semantics of the attribute qualifiers may vary according to the type of input record. The position attribute, for example, refers to the position of the field within the record for delimited records while, for fixed-length records, it refers to starting position of the field relative to the beginning of the record.

Supported Qualifiers

The following table lists the qualifiers that are available when constructing embedded field descriptors in your SQL statements.

Qualifier Keyword	Argument	Description	Input/Output
external=XXX (in association with size attribute)	XXX may be INT or FLOAT For INT, size must be 1, 2, 4, or 8 for tiny, short, int or long integers For FLOAT, size must be 4 or 8 for short float or long float respectively	Use when the data item is (or is to be delivered) in an external numeric format. It is valid only for numeric data. External formats may not be portable across all platforms.	Both, but only for fixed-length record formats
dateform=N (may be associated with external and/or size attributes: if so, N must be consistent with both)	N is the number constant to which the dateform is mapped. The number constants and the applicable dateforms are described in the topic Date/Time Data 3	The format of the input date or date/time, for parsing purposes; or the literal form in which the output date or date/time is to be written. Fields that are to be read from or written to an external (binary) date/time format are applicable only to fixed-length records.	Both
size=N	N is an integer	Size of the field, in bytes	Both, but not valid for delimited data

position=N	N is a number	1-base starting position applicable to the field in the input or output record: • for fixed-length records, it is the offset of the start of the field from the beginning of the record • for delimited records, it is number of the field in the field order	Both

Generated Fields

enum	No argument	Generates a value of smallint, integer or BigInt type that is a row count. For input it must be the only attribute in the embedded element, i.e. {ENUM}.	Both
sysdate	No argument	Generates a value that reads the system date and time. • For output it is a literal string whose format determined by an accompanying dateform specification. • For input it is a TIMESTAMP and it must be the only attribute in the embedded element, i.e. {SYSDATE}. It must be mapped to an SQL column of the appropriate date/time type. It can be cast if necessary.	Both

BLOB-related Attributes	Three attributes—ascii_blob, blob_size amd blob_stream—apply only to input that is destined for writing to BLOB columns. For details, refer to the topic BLOB Inputs 9.

2.1 Field Mapping

Position and Size of Fields

If the data is fixed position or delimited without using data delimiters, then specifications must be provided to describe where the field is in the record, its size and, in some cases, its format.

The position attribute allows fields in delimited files to be out of order with respect to the input or output statement.

- For input, it also facilitates field skipping and re-use of a single input field to fill more than one database column.
- For output, all field positions must be specified exactly one time. If you need to pass the same data to more than one output field then use aliased columns in your SELECT statement to include them in the output set.

For Input Records

For an import, the position attribute may be omitted for all fields if the input record format is delimited using a separator AND the mapping order of the input fields to the database columns is identical.

> If a generated field {ENUM} or {SYSDATE} is included, it does not necessarily break implicit ordering.

Otherwise, position is required in all cases.

> Note that you must not have a mix where some fields have a position and some do not. It must be all or nothing.

The attribute specifications are interpolated after each column identifier in the DML statement. For example:

```
UPDATE OR INSERT INTO COUNTRY (COUNTRY {position=1 size=15},
  CURRENCY {position=16 size=10}) VALUES (?,?)


EXECUTE PROCEDURE ADD_EMP_PROJ (?{position=1 size=2}, ?{position=2 size=5})
```

The position attribute has different semantics according to whether the input record format is delimited or fixed:

- in a delimited record format, position matches field position from left to right, so {position=1} maps the first field, {position=2} the second field, and so on.
- in a fixed record format, position matches the position in the record where the field starts, so position 1 is the first byte of the first field (or, in this case, the first variable). In this example, the size of the first variable is 2 bytes, so the second field is at position 3. The {size=} attribute is not used in fixed format descriptors.

> In the EXECUTE PROCEDURE example above, the input fields would overlap if the input records were fixed format. Overlapping of imported fields is not disallowed so, if you have a good reason to do it, you can.

For Output Records

On exported records, all fields, including filler fields, must be specified on output exactly once. No overlapping is allowed between either SQL output fields or filler fields. For output fields retrieved from the SQL query, an example for output to a fixed-length record might be expressed as follows:

```
options dateform=17 hyphenateDate # dd-mm-yy hh:mm:ss
SELECT
  EMP_NO {position=1 size=2},
  FIRST_NAME {position=3 size=15},
```

```
  LAST_NAME {position=18 size=20},
  PHONE_EXT {position=38 size=4},
  HIRE_DATE {position=38 size=19}
FROM EMPLOYEE
```

Text Columns

For CHAR or VARCHAR columns going to fixed output, right-padding with spaces will be performed without the need to describe the attributes of the fields.

Re-mapping Field Position

The default position for delimited data into or out of SQL with no special field positions is the order of their occurrence in the SQL statement. For delimited records, if the field order of the input or output record is different to the SQL statement order, the `{position=}` element can optionally be used to re-map the order in which the fields are read from or written to the file.

Multiple Usage of Input Fields

For reading input from a delimited record, the `{position=N}` phrase can be used more than once where you want to write the same input field to multiple columns in the destination table.

Alignment Issues

dbFile does not use the input file alignment as all fields use an intermediate area which is aligned for the worst case. However, given that it is in the order of things for numbers of all types to require alignment, potential exists for it to be a problem for the program processing an exported file.

Bad Descriptors

If you provide field descriptors that do not fit (wrong size or wrongly qualified for type) for either input or output, a parser error will occur and the run will be aborted. For filler fields 11 there is some margin for error, provided the specifications allow for a size that is large enough.

2.2 Blob Inputs

> Since the size of data in BLOBs is, by design, unspecified, it is practicable only to support them in imports. The input file is expected to include a field corresponding to the BLOB column in the DSQL statement, containing either the data that is to be written or an absolute or relative path to a file containing the data for that record.

The attributes for the field are listed in a curly-bracketed embedded descriptor 5, along with any other attributes that are valid. For example:

```
INSERT INTO ATABLE (
  ...,
  MY_BLOB_COLUMN {position=4, ascii_blob},
  ...
                    )
values (..., ?, ...)
```

> You may specify only one set of attributes for the BLOB input, so never try to load two or more BLOBs of different types for the same column.

Blob-related Attributes

Qualifier Keyword	Argument	Equiv. Switch	Description	Input/Output
ascii_blob %S	No argument %S is not a structural element in the syntax. It represents the data that is in the input field at the given position.		The parser will try to use %S as the absolute or relative path to a file and, if it finds the file, reads its data into the variable it has assigned for the BLOB column. If it fails to find the file, it uses the bytes of %S as the data for the BLOB. Don't be alarmed by use of the "ascii" moniker in this qualifier. It was used by the original author of the software as a synonym for "text". You should ensure that your text file or inline data uses a character set that is compatible with that of the destination text BLOB.	Input
blob_size=N	N is the segment size	-b N	Unless instructed otherwise, dbFile takes segmented binary as the default BLOB style and reads in the data as binary segments of N bytes. The default segment size is 80 bytes. Don't worry if the segment size set here is not the same as that of the target column. Even though dbFile may attempt to slice up the text into N-sized segments, the size of BLOB segments arriving at the Firebird server is irrelevant to how data is segmented for storage: Firebird always uses its own algorithms.	Input
NB :: associated command line switch that has no equivalent options attribute		-a	Include this switch in the command line to instruct dbFile to read line-break characters as segment boundaries, according to the argument set by the global option `lineend=C` or switch `-l C` (where C is one of M, P or U for the platform-specific line-break convention used on traditional Macintosh, Windows or POSIX, respectively).	Input
blob_stream	No argument	-r	Use blob streaming and ignore segments. The blob_size and blob_stream attributes are incompatible. However, if you accidentally supply both then dbFile will work with the directive it encounters last.	Input

2.3 Filler Fields for Exports

For specifying output records ONLY, the directive `options filler={list_of_attributes}` can used as often as needed to describe an extra "filler" field, i.e. one not derived from the SELECT query, in the layout of the record. The rules work differently according to whether you are specifying the output as fixed length or delimited records.

There is no way to pass these descriptors from the command line: they must be on an options line in a control file.

Specifying Filler Fields

Each filler entry describes one field. The list_of_attributes consists of four arguments in a pre-defined order and separated by semi-colons. The arguments are not case-sensitive. The format is:

```
options filler=s;e;t;f
```

The variables=N option is required if one or more filler entries are present.

Filler Attribute List Arguments

Argument	Position	Usage
s	First	s is a number specifying the starting position • for fixed-length records, s is 1 for the first character of the first field. • for delimited records, s is 1 for the first field
e	Second	e is a number: • for fixed-length records, e specifies the ending position for the field in the record. The size of the fixed-length field is thus (e - s + 1). • for delimited records, an e value is used only if the t argument is used, in which case it expresses the size of the value specified by the t argument SIZE MATTERS! When interpolating non-database fields into output records, overlaps will cause the process to stop and complain. Take care that you allow enough space to accommodate the maximum possible size of the generated data.
t	Third	t can be one of three things: a) a single character specifying the fill character to be used to fill the field. Suggested characters are blank, zero, or dash. (For blanks,

leaving the argument empty is equivalent to typing in a space character.)

b) the keyword SYSDATE to output the host server date or date/time, supported by a valid date format attribute specified as the f argument

c) the keyword ENUM to output the value as a record counter (generated as (record count + startcount)) specified by a valid number format attribute as the f argument

f Last f is a number or string constant for the external format of the field where a keyword was specified for the t argument. It is ignored if the t argument was used to specify a filler character.

If t=SYSDATE, the number constants and templates for the range of date formats are explained in the section Date/Time Data 3.

If t=ENUM, the following string constants are valid:

- FmtInt16 (binary, size 2 – if less than 32,768 records will be selected)
- FmtInt32 (binary, size 4)
- FmtInt64 (binary, size 8)
- FmtCstring (for delimited file)
- FmtFixString (for fixed file – size must be large enough for the number of digits expected)

Overlapping Fields

Use this option as many times as you have extra fields to include but take care that your specifications do not cause fields to overlap one another.

- If you actually intend for input fields to overlap, dbFile allows it
- If you try to overlap output fields, the process will be aborted

Representing Null Arguments

All four arguments are required in the list_of_attributes. Where an argument is null, omit the value but include the separator. For example, the following directive will work fine to put :

```
options filler=4;;sysdate;18
```

3 Using a Control File

Kicking off a file import or export with dbFile involves passing a number of switches and arguments to the executable program. For simple, one-off operations, that can be quick and handy. Even so, even the most minimal command line will include specifications for direction (import or export), the database path, the file specification for the import or export file and Firebird authentication credentials.

For anything you might want to repeat, or for an operation with a lot more than the minimum options, you can prepare a control file in your favourite text editor. Supposing you had created a properly constructed control file named mydef.def, specifying the options for exporting data from a Firebird database, the following command is all that is required:

```
dbFile -x mydef.def
```

The -x switch, while available on POSIX platforms, is optional there. If it is omitted, the file is read via the `stdin` stream.

Writing a Control File

A control file comprises a set of options and an SQL INSERT statement. Options are specified in the control file using the marker keyword options with a range of keywords that equate to the corresponding command-line switches. You can mix options in the control file and the command line if you wish. Options specified in the control file overrule any switches passed in the command line.

The control file may be any number of lines. Options lines come first: each line can be up to 4096 characters long. One and only one DSQL statement follows, consisting of any number of lines, with a maximum size of 4088 characters, including white space, embedded data qualifying attributes and commented sections.

For an import, the control file may be written so as to embed the data for input. by including in a section after the SQL statement, headed up by a STARTDATA directive 19.

The *options* Keyword

The syntax for expressing an option is:

```
options <keyword>[=<argument>]
```

where <keyword> is one of a set of available keywords and <argument> is a valid argument for the specific keyword. The <, > and square brace symbols used here are not part of the syntax.

Arguments are not quoted, even if they are strings. Space characters should not be inserted on either side of the "=" sign.

Some options do not require an argument. For example, in `options excel`, the keyword excel stands alone to instruct the parser to conform to certain specific rules that apply to reading and writing .csv files generated by or for Microsoft Excel. In general, for options that do take arguments, the argument is required.

Multiple options in a control file can be continuously "run-on" with no line breaks or can be broken up into separate lines to simplify editing. However, take note of some special rules that apply to option lines:

- If you use line breaks between options, each option line must begin with the option keyword.
- If line breaks are not used, use the option keyword just once, at the beginning of the line, and separate each option from the next by one space character.
- You can mix multi-option and single-option lines in your control file.

Minimum Options

The minimum options are:

Keyword	Purpose
charset=	Sets the client character set. Its argument is the identifier or alias of a character set known to the database
direction=	Indicates whether the task is an import (i) or an export (o)
db=	An unquoted string that is the full connection string (server and file path or alias) to the Firebird database
file=	An unquoted string that is the filesystem path to the input or output file that is to be read from or written to
user=	Firebird login user name. Can be omitted if the ISC_USER and ISC_PASSWORD environment variables are set
passwd=	Firebird login password. Can be omitted if the ISC_USER and ISC_PASSWORD environment variables are set
delimiter=	Required for delimited text input or output, this option takes as its argument the character that is used as the field delimiter. Refer to the notes in the introduction regarding delimiter usage[1].
variables=	An integer indicating the number of parameters in your INSERT statement

Login Credentials

Both the user name and password must be supplied. If the ISC_USER and ISC_PASSWORD environment variables are available, the user name and password can be omitted.

NOTE :: Mixing will NOT work. That is to say, if you supply just one of user name or password, and rely on an environment variable to complete the credentials, you'll get an authentication error.

The SQL Statement

The SQL statement follows all of the options lines.

- For imports, dbFile works with parameterised DSQL statements UPDATE, INSERT, UPDATE

OR INSERT and with calls to executable stored procedures.

- For exports, dbfile retrieves data with a SELECT statement which may be parameterised. It can be any valid SELECT statement, including joins, subqueries, views or SELECTs on selectable stored procedures.

The statement can occupy any number of lines in the control file. No special continuation characters are needed to continue a statement but words may not be split between lines as a space will always be added in between multiple lines of SQL.

Embedded Comments

The "pound" (#) sign may be used on any option or SQL line to indicate that all characters to its right are to be ignored. Parsing will recommence at the start of the next line.

> CAUTION
> Specifying the pound sign as a field delimiter, viz., option delimiter=# is not recommended, since the parser will interpret the # marker of your comment as a delimiter character and the rest of the line will continue to be parsed until the next # character.

Parameters and Position

Firebird parameter placeholders are positional, each represented by a questionmark (?), for example:

```
UPDATE OR INSERT INTO COUNTRY (COUNTRY,CURRENCY) VALUES (?,?)
```

For cases where the field order in the input file is different to that in the DSQL statement, an optional, 1-based position argument can be included in the statement to map the field position of each field of the file record to the field position in the statement's input list.

The following example (normally unnecessary) expresses exact mapping for the statement above, where the input records have fields in the same order as the input list of the DSQL statement:

```
UPDATE OR INSERT INTO COUNTRY (
  COUNTRY {position=1},
  CURRENCY {position=2})
  VALUES (?,?)
```

The following syntax enforces the mapping of the second field of the input record to the first parameter of the INSERT statement:

```
UPDATE OR INSERT INTO COUNTRY (
  CURRENCY {position=2},COUNTRY {position=1})
  VALUES (?,?)
```

> NOTE :: This specific usage of the {position} specifier is for delimited record types. Fixed format records use it differently, as explained below under Fixed Position Text Records.

Processing the Control File

Processing the control file is no more than invoking the dbFile executable with the switch `-x <path_to_control_file>`

For an example with simplified syntax, to execute a file named mydef.def in which you have set your options, call

```
dbfile -x mydef.def
```

In practice, of course you must abide by platform and shell rules for invoking executables from the command line and filesystem rules for accessing and expressing the path to the control file.

3.1 Import Examples

The following examples illustrate importing data from a variety of formats into the Country table in the sample Employee database (located in $firebird/firebird/examples/empbuild/employee.fdb in a typical Firebird installation).

Delimited Text

In a delimited text file, data records are separated by line breaks, while fields within each record are separated consistently with a single 7-byte character, such as a TAB character or a rarely-used printable character, such as the "pipe" symbol (|, ASCII 124/x07C).

Your delimited text data file might contain data like this:

```
USA|Dollar
England|Pound
Canada|CdnDlr
Switzerland|SFranc
Japan|Yen
Italy|Euro
France|Euro
Germany|Euro
Australia|ADollar
Hong Kong|HKDollar
Netherlands|Euro
Belgium|Euro
Austria|Euro
Fiji|FDollar
```

A control file for the import could be:

```
options  db=127.0.0.1:/opt/firebird/examples/empbuild/employee.fdb
options file=countries.txt
options direction=i
options passwd=masterkey
options user=SYSDBA
options variables=2
options delimiter=|
UPDATE OR INSERT INTO COUNTRY (COUNTRY,CURRENCY) VALUES (?,?)
```

.CSV Format

OpenOffice, Microsoft Excel and many other data storage applications can export data as plain text in comma-separated values ("CSV") format. The characteristics of CSV include double-quoting on string values and fields separated by commas or some other configurable separator character.

For example, an input file for our COUNTRY table, using a semicolon as the field separator, might look like this:

```
"COUNTRY";"CURRENCY"
"USA";"Dollar"
"England";"Pound"
"Canada";"CdnDlr"
"Switzerland";"SFranc"
"Japan";"Yen"
```

```
"Italy";"Euro"
"France";"Euro"
"Germany";"Euro"
"Australia";"ADollar"
"Hong Kong";"HKDollar"
"Netherlands";"Euro"
"Belgium";"Euro"
"Austria";"Euro"
"Fiji";"FDollar"
```

The control file could be :

```
options excel quote=" delimiter=; direction=i file=countries.txt
options variables=2 user=SYSDBA passwd=masterkey
options db=127.0.0.1:employee
UPDATE OR INSERT INTO COUNTRY (COUNTRY,CURRENCY) VALUES (?,?)
```

Fixed Position Text Records

In a fixed-length format, there are no fields of variable length and no delimiters. Interpretation of this input data as fields and records depends on two attributes:

1. The starting position of each field
2. The size of each field (number of characters)

Position and Size Arguments

A specialised usage of the embedded directive descriptors 5 specifies these two required attributes to dbFile for each field in the fixed position text record. The interpolated clause (one for each input field) has the form

```
{position=p size=s}
```

where p is the 1-based start position in the record and s is the maximum number of characters in the field. For example, a record with the form

```
Switzerland    SFranc
```

has two fields, the first defined by `{position=1 size=15}`, the second by `{position=16 size=10}` .

Suppose your fixed position countries.txt file has fixed position data such as the following:

```
USA            Dollar
England        Pound
Canada         CdnDlr
Switzerland    SFranc
Japan          Yen
Italy          Euro
France         Euro
Germany        Euro
Australia      ADollar
Hong Kong      HKDollar
Netherlands    Euro
Belgium        Euro
Austria        Euro
Fiji           FDollar
```

A control file for this import could be:

```
options direction=i file=countries.txt variables=2
```

```
options user=SYSDBA passwd=masterkey db=127.0.0.1:employee
UPDATE OR INSERT INTO COUNTRY (COUNTRY {position=1 size=15},
  CURRENCY {position=16 size=10}) VALUES (?,?)
```

The `variables=` option specified here indicates that the DSQL statement has two parameters.

Fixed Position Data without Line Breaks

Certain fixed position data formats do not separate records by line breaks: data just runs in a continuous string. An additional option keyword is available for specifying the record length for such formats:

```
lrecl=nn
```

where nn is length of one record.

> For fixed position input in certain character sets and/or file formats, nn could be the byte count, rather than the character count. If you have non-ANSI character sets or platform-specific file formats in the picture, experimentation in a test environment is strongly recommended, to establish the specifics of the case.

3.2 Export Example

With dbfile you can export data into a delimited or fixed size file.

The syntax is very similar to that for importing. The direction option changes to direction=o and the SQL statement becomes a SELECT query over one or more tables or views or a selectable stored procedure that returns rows.

A control file consisting of the following:

```
options direction=o file=countries.txt variables=2 user=SYSDBA
    passwd=masterkey db=127.0.0.1:employee
SELECT COUNTRY {position=1 size=15},
    CURRENCY {position=16 size=10}) FROM COUNTRY
```

will produce the following fixed size data file:

```
USA             Dollar
England         Pound
Canada          CdnDlr
Switzerland     SFranc
Japan           Yen
Italy           Euro
France          Euro
Germany         Euro
Australia       ADollar
Hong Kong       HKDollar
Netherlands     Euro
Belgium         Euro
Austria         Euro
Fiji            FDollar
```

> The named output file is created by dbFile. If a file of that name exists already, dbFile will overwrite it.

3.3 The STARTDATA Directive

For those who want it, the data records for an import of delimited records can be optionally embedded into the control file. The directive STARTDATA following the SQL statement tells dbFile to read all content from the beginning of the next line to the end-of-file marker as input.

Limitations

Although embedding the input data in the control file is supported, it is not recommended: import files are usually outputs from other sources of data and there are strong reasons to keep data and processing directives cleanly separated. Moreover, it restricts your control file to a one-off usage, defeating the objective of storing a repeatable process conveniently in a file.

If you are determined to use STARTDATA, keep in mind that there are limitations to this usage:

- It is invalid for data in Excel format data since the control file itself is not compatible with Excel
- It does not support input of fixed record format

On Windows, bear in mind that conditions exist where the operating system does not correctly write an end-of-file marker if the last line of data is not completed with a hard "paragraph break" (carriage return + linefeed). It is a good precaution therefore to make a point of hitting the Enter key at the end of your block of data, before saving the file.

Usage in a Raw Command

On POSIX, it is possible to invoke dbFile and import data without using a control file. It is possible (albeit not particularly practicable!) to append a STARTDATA block to a raw dbFile command using regular shell syntax. For more information, see the examples in the topic Using the Command Line Without a Control File 22.

3.4 Reserved Words

The words OPTIONS and STARTDATA—whether in upper or lower case or as quoted identifiers—are reserved for dbFile and may not be used in the SQL statements that you pass to the application. Ideally, if you know at design time that you need to use dbFile to load or unload data, it is advisable to avoid creating columns that have these names.

If columns already exist that use these names, you may use ALTER TABLE statements to rename the problem columns and add COMPUTED BY columns with the original names to duplicate the data from the renamed ones.

NOTE :: an alternative is to retain the original columns and create the computed columns for dbFile operations. However, for loading data, you will need to write BEFORE INSERT OR UPDATE triggers to ensure that the "Attempted update of read-only column" exception (code 335544359, gdscode read_only_field) is caught and handled by diverting the write to the real column.

3.5 More Useful Options

The following list includes some more options that you can use in control files to augment and prefect your data import or export. The full list of keywords and switches

25 is at the end of this document.

Keyword	Argument[s]	Purpose
discards=NAME	NAME is the relative or fully-specified path to a file	Discards from input processing can be sent to a file. On POSIX, the default is /dev/null.
errors=N	Stop loading after N errors	Causes the import or export operation to stop after N errors have occurred. Default is 1 (stop on the first error).
	Note also that the keyword short_ok makes a "short" input record into a warning condition rather than throwing an error.	
excel	No arguments	Informs dbFile that the file is in Excel format. For input files dbFile will ignore the first line. For output files, it will generate column names as the first line of the file.
file=NAME	NAME is the relative or fully-specified path to the input or output file	The named output file is created by dbFile. If a file of that name exists already, dbFile will overwrite it.
fractComma	No arguments	Fractional numbers use a comma as the decimal point. The default decimal separator is a period.
hyphenateDate	No arguments	Output dates will use hyphens as separators, instead of the default slashes.
lineend=x	x = M, P or U	Indicates the line-break type (M for Mac, P for Windows/DOS, U for Unix). The default is the standard line-break type applicable to the operating system.
logfile=NAME	NAME is a relative or fully-specified path to a file	File for logging both stdout and stderr. On Windows, if logging is wanted, this is the only way to get it.
lrecl=N	N is the length of a fixed-length input record, where there are no line breaks separating records	Used only for fixed size records only and only those with with no line endings (like some IBM output formats). There is a caveat: there can be no short records, except the last, and applying the wrong value to N will

		create data in your database that is wrecked beyond repair.
pad=C	C is a single character to be used as padding	For fixed output, use the character specified by C to fill in any parts of the output record not described by either SQL output or filler fields. Escaping 25 is valid and may be required if your pad character is one that has semantic significance in your command shell.
quote=X	X must be either a single or double quote character	For use only if the delimiter option is used. Specifies the style of quote used for quoting strings. Strings only need to be quoted if the delimiter character potentially may occur in the string. If the quote option is specified, all output strings will be quoted ALWAYS.
	Numbers do not need to be quoted and will never appear in output with quotes.	
rows=N	N is a number	Stop the import or export after after N rows have been processed. Default: no limit on the number of rows to process.
short_ok	No arguments	The default is to treat short input records as errors and not to load them (skip them and optionally log them). The effect of short_ok is to specify that short records are to be loaded, with missing fields treated as null.
skip=N	N is a number	Skips N records in the input file or the SQL query output before commencing the import or export process.
startcount=N	N is a number	Uses N as the base for generating the filler count that is specified by defining the filler field ENUM. The output value is generated as (record count so far + startcount). The default value for N is 0.
trans=N	N is a number	Commit the transaction after every N rows of input. Default: 5000.
verbose	No arguments	Execute the process in verbose mode. The default is to report only errors.

See full List of Options and Switches 25

4 Using the Command Line

> Of course, dbFile is a command-line utility: that is, it is designed to be run from a command shell. Invoking dbFile with the -x switch pointing to a well-thought-out control file is the most natural and sturdy way to perform your import or export. If you are operating on a Windows platform, it is the only way to use dbFile.

But, even if your platform is not Windows, the control file approach is recommended, especially if you plan to repeat similar operations. As explained earlier, the typing at run-time is minimal and you have a much better chance of specifying the operation correctly and completely.

Using the Command Line Without a Control File

On POSIX platforms, including MacOSX, an import from or export to a delimited file may be done directly from the command line, without using a control file. Those who habitually write scripts to accomplish tasks on these platforms will find equivalents to virtually all of the descriptor options available for control file usage. As well as each of the minimum options described for usage in control file mode, dbFile has switches to enable you to pass the attributes applicable to delimited file imports or exports. They are described in their entirety, along with all of the control file options, in the next section, List of Options and Switches 25.

Command Format

The format for the "raw" invocation of dbFile is the same as when invoking it with a control file except that, while there is no -x switch, there will be a number of other switches, along with their arguments. Following the switches is a block of redirected data wrapped in a pair of alphanumeric symbols invented by you (avoiding any symbols that are significant to the shell parser, of course!). For example, you could use the characters 'EOF' or 'XYZ' as the marker for your block. The dots in the following patterns are standing in for your actual content.

```
dbfile ....<<EOF
...content...
EOF


dbfile ....<<XYZ
...content...
XYZ
```

What you include in your block is up to you. In the simplest case it would be just your SQL SELECT or

DML statement; but it could also include options directives (ahead of the SQL) and even a STARTDATA block 19 following the SQL statement. The same rules of placement that apply inside a control file 13 also apply inside your redirected block.

The Essential Switches

The "essential" switches are the ones that are required to specify a non-control import or export from or to a delimited file. Each consists of an alphabetic character preceded by a hyphen, e.g. -u, followed by a space followed (in most cases) by an argument. For example:

```
dbfile -u sparky -p icuryy4me
```

Switches can be lower or upper case on any platform; that is, a lower case letter and its ANSI upper case equivalent are semantically identical for switches. The same may or may not be true for the argument! Arguments are unquoted.

User Authentication Switches

Switch	Argument	Comments	Options Equivalent
-u	User name	Valid Firebird user name. The user must be one that has the appropriate privileges to the database tables and/or objects that will be accessed by the DSQL statement. Case-insensitive unless the user was created in double-quotes AND has any characters that are not 7-bit upper-case.	USER=
-p	Password	Valid password for the user name. Always case-sensitive.	PASSWORD=
		If the environment variables ISC_USER and ISC_PASSWORD are available, or you are on a POSIX server logged in as root, then these switches are optional. On a POSIX server where system UIDs have authentication profiles in the Firebird security database, the system user login should be sufficient.	

Data-related Switches

Switch	Argument	Comments	Options Equivalent
-db	A string	Full qualified path or alias to the database, according to platform, server model and whether it is local or remote.	DB=

Don't forget that, on POSIX, a local connection cannot be made if the server is Superserver and, on Windows, Classic 1.5.x versions cannot take a local connection.

-i or -o	A file specification (string)	The argument is the qualified or relative path to the input file if the switch is -i or the output file if the switch is -o.	DIRECTION= together with FILE=
-c	Single character or xNN	Character that is used as the field delimiter in the input record or is to be used as the field delimiter in the output record. You may use xNN hex notation for special characters; e.g. x08 for tab	DELIMITER=
-n	A whole number	Number of variables in the input or output record. For output it may be a '?' but it should be escaped 25 to ensure it is ignored by the parser.	VARIABLES=
-t	A character set identifier	Sets the client character set. NAME is the identifier or alias of a character set known to the database	CHARSET=

More switches are described in the next section 25.

For delimited files, using the command line alone may be sufficient. However, note that

- unless the -c and -n switches are used for delimited file input, a control file must be used to specify options.
- all options specified in a control file override any equivalent arguments specified in the command line call

The SQL Statement

The SQL statement should be the first (or only) content in the redirected block. If you happen to incorporate the actual data in your command, the STARTDATA directive must follow the SQL statement on a line of its own.

The following illustrates a "quick and dirty" call from the command line on POSIX to load a trivial set of data on the fly:

```
echo "let us try some silly load stuff"
dbFile -n 2 -i -d /tmp/mydb.fdb ..... -c '|' <<EOF
insert into fubar (foo, bar) values (? ?)
STARTDATA
monkey|shine
blue|moon
sick|input
crazy|text
EOF
echo "we just loaded some silly data"
```

Special Characters

Characters such as brackets, semi-colon, colon, comma, parentheses, single or double quotes, questionmark or the pipe symbol (vertical bar) may have to be escaped on the command line, if the same characters are syntactic elements of the command shell interpreter.

Escaping Shell-Sensitive Characters

"Escaping a character" means inserting another special character, before the shell-sensitive character to tell the command shell interpreter to ignore its syntactic significance and treat it as a literal. The inserted character and the character it operates on are together referred to as "an escape sequence".

SQL users may be already familiar with "escaping" literal single-quote characters by preceding them with another single quote, a convention sometimes referred to as "doubling". The concept is similar, but command shells do not use doubling. Instead, they use a specific character that is reserved as the escape character.

- In most POSIX shells, including MacOSX, the escape character is the backslash (\)
- In the Windows command shell the escape character is the caret (^)

Line Break Characters

Both carriage return (CR) and newline (NL) alias linefeed (LF) are very special characters. One or the other, or both, are used as line terminators. Which one depends on the operating environment. On Windows, a line break consists of both CR and LF, in that order. On POSIX, including the MacOSX text terminal, it is a NL alone. In the original Macintosh operating system it was CR alone. On a MacOSX system, the GUI terminal might be configured to conform with the text command shell or with other GUI applications, which still use the native Macintosh convention.

Delimited Input Files

Since string data might contain embedded line terminators of either kind, there can be problems for output as well as for input files.Text blob files are handled correctly, but delimited input files with embedded line breaks will cause fatal problems.

> Be aware that, for delimited input files, dbFile does not have a catch-all solution for the problem of embedded line breaks. It remains the user's responsibility to deal with any pre-processing of the data to hide the line-breaks.

Fixed format data files are not prone to this problem since a different read method is used, that ignores the syntactic role of CR and NL altogether.

Delimited Output Files

When outputting delimited files, dbFile substitutes all embedded NL/LF and CR characters with the non-printable 7-bit characters x01 and x02, respectively. Such output can be passed safely in an input file and dbFile will replace them appropriately with the NL/LF and CR characters.

5 List of Options and Switches

The following is the full list of options available, along with the corresponding command-line switches that could be used instead, if not provided in a control file.

Keyword	Argument[s]	Switch Equivalent	Purpose

charset=NAME	NAME is an identifier	-t NAME	Sets the client character set. NAME is the identifier or alias of a character set known to the database
dateform=N	N is a number	-f N	N identifies the external date or date/time format of input or output date literals. The numbers and the corresponding format templates are listed in the topic Date/Time Data 3.
db=ARG	ARG is an unquoted string that is the fully qualified path or alias to the database, according to platform, server model and whether it is local or remote.	-d ARG	Targets the database to which dbFile must connect in order to write or read the data.

Don't forget that, on POSIX, a local connection cannot be made if the server is Superserver and, on Windows, Classic 1.5.x versions cannot take a local connection.

delimiter=C or delimiter=xNN	C is an ASCII character, which may be in hex notation as xNN for special characters; e.g., x08 for TAB	-c C	Required for delimited text input or output, it takes as its argument the character that is used as the field delimiter. Refer to the notes in the introduction regarding delimiter usage 1.
direction=C	In a control file, C is a character which must be either i or o For a switch, the argument NAME is a string which is the fully qualified or relative path to the data file	-i NAME or -o NAME	Specifies whether the operation is an import or an export. • In a control file, i specifies an import, o an export. The input or output file is specified in a separate options attribute, `file=NAME` • On a command-line, the -i or -o switch takes NAME as its argument.

The direction= directive may be omitted if you use the STARTDATA feature to embed the input data in your control file.

discards=NAME	NAME is the relative or fully-specified path to a file		Discards from input processing can be sent to a file. On POSIX, the default is /dev/null.
errors=N	Stop loading after N errors		Causes the import or export operation to stop after N errors have occurred. Default is 1 (stop on the first error).

excel	No arguments	-e	Informs dbFile that the file is in Excel format. For input files dbFile will ignore the first line. For output files, it will generate column names as the first line of the file.
file=NAME	NAME is the relative or fully-specified path to the input or output file	Argument to the -i or -o switch	N.B. the control file can also be the input file - see the topic about the STARTDATA directive 19.
fractComma	No arguments	-m	Fractional numbers use a comma as the decimal point. The default decimal separator is a period.
hyphenateDate	No arguments	-h	Use hyphens instead of the default slashes as element separators for output dates.
lineend=C	C is a character, one of M, P or U	-l C	Indicates the line-break type (M for Mac, P for Windows/DOS, U for Unix). The default is the standard line-break type applicable to the operating system.
logfile=NAME	NAME is a relative or fully-specified file path	-s NAME	Path to the file name for logging both stdout and stderr. On POSIX, stdout and stderr are the default. On Windows, no logging will be provided unless the logfile option or -s switch is explicitly supplied.
lrecl=N	N is the length of a fixed-length input record, where there are no line breaks separating records		Used only for fixed size records only and only those with with no line endings (like some IBM output formats). There is a caveat: there can be no short records, except the last, and applying the wrong value to N will create data in your database that is wrecked beyond repair.
pad=C	C is a single character to be used as padding		For fixed output, use the character specified by C to fill in any parts of the output record not described by either SQL output or filler fields. Escaping is valid. For information about escape characters, refer to the topic Escaping Shell-sensitive Characters 25.
passwd=NAME	NAME is an unquoted string. It is case-sensitive	-p NAME	Firebird login user name. Can be omitted if the ISC_USER and ISC_PASSWORD environment variables are set

quote=X	X must be either a single or double quote character NOTE :: If you need to specify single quotes then only the control file method supports it.	-q Switch takes no argument: only double-quotes are available Valid only if the -c switch is used	For use only if the delimiter option is used. Specifies the style of quote used for quoting strings. Strings only need to be quoted if the delimiter character potentially may occur in the string. If the quote option is specified, all output strings will be quoted ALWAYS. Numbers do not need to be quoted and will never appear in output with quotes.
rows=N	N is a number		Stop the import or export after after N rows have been processed. Default: no limit on the number of rows to process.
short_ok	No arguments	-z	The default is to treat short input records as errors and not to load them (skip them and optionally log them). The effect of short_ok is to specify that short records are to be loaded, with missing fields treated as null.
skip=N	N is a number		Skips N records in the input file or the SQL query output before commencing the import or export process.
startcount=N	N is a number	N	Uses N as the base for generated filler counts. As each record is processed, N is incremented by 1. The default base N is 0.
timestamp	No arguments		Include time in generated date/time fields (default is just date)
trans=N	N is a number		Commit the transaction after every N rows of input. Default: 5000. The minimum for N is 100, maximum 20000.
user=NAME	NAME is a string. If the user was created in double-quotes AND has any characters that are not 7-bit upper-case then it must be double-quoted and	-u NAME	Firebird login user name. Can be omitted if the ISC_USER and ISC_PASSWORD environment variables are set. The user must be one that has the appropriate privileges to the database tables and/or objects that will be accessed by the DSQL statement.

	quote symbols may need to be escaped.		
variables=N	N is a number, viz., the number of input or output parameters	-n N	For input this directive is required. For output, you may omit it if there are no filler fields. If provided, it is checked against columns specified in the query.

For output, if you have a SELECT * query as your SQL, you may still include a `variables=` descriptor. If N is unknown in these circumstances, use an escaped questionmark (\?) for the N argument.

verbose	No arguments	-v	Execute the process in verbose mode. The default is to report only errors.
xtra_fields=N	N is a number		A count of the extra "filler" fields to be built for output. For information about filler fields, see the topic Filler Fields for Exports 11.
year=NN	NN is a number representing the last 2 digits of a year	-y NN	Treat any 2-digit year in input dates after this number as belonging to the previous century

Index

- S -

- T -

- U -

- W -