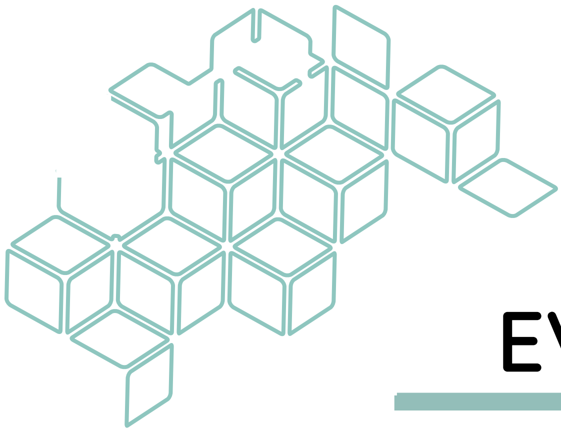


EVL - QVD Utils

version 2.7

team@evltool.com https://www.evltool.com/evl-qvd-utils

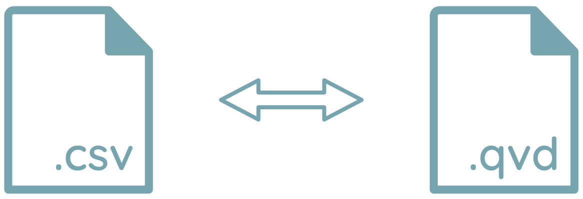
This manual is for **QVD Utils** (version 2.8), an EVL Microservice which read QVD, write QVD/QVX and get information from QVD/QVX (Qlik's file formats). Using QVD Utils standalone, i.e. without other EVL Microservices, allows only to read/write QVD files from/into CSV. But together with other EVL Microservices, or with full EVL installation, you can transform Qlik QVD files also from/into many other formats or to read/write directly from/into various databases.

Table of Contents

1 Introduction

QVD Utils enables reading/writing QVD files without a need to use Qlik Sense or QlikView. Also provides metadata from QVD/QVX header. It is useful when you need to convert various sources directly to/from a QVD file and you don't want to overload your Qlik server by processing many, or large, QVD files.

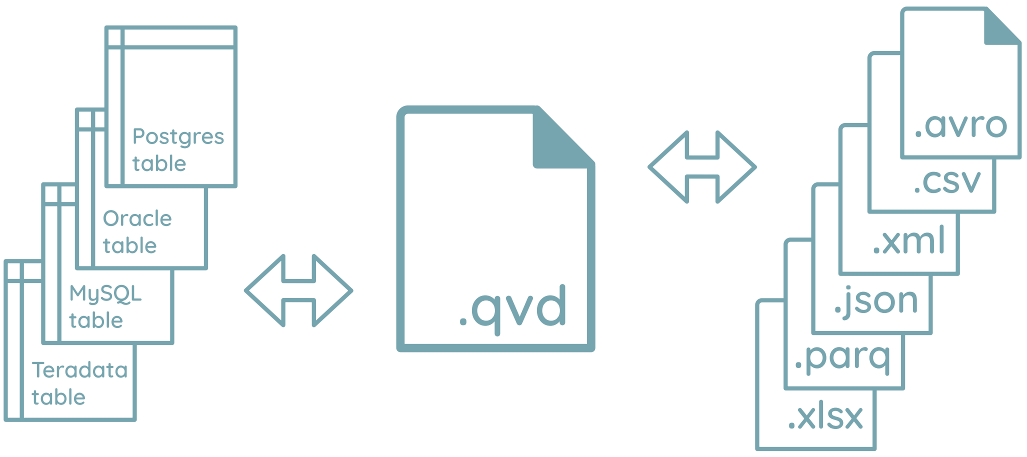
1.1 QVD Utils used standalone



Having QVD Utils without any other EVL Microservice contain these scripts:

- Section 4.1 [csv2evd], page 6, – generate EVL data definition file (EVD file) based on CSV
- Section 4.2 [csv2qvd], page 8, – convert CSV to QVD
- Section 4.3 [qvd2csv], page 11, – convert QVD to CSV
- Section 4.4 [qvd2evd], page 14, – generate EVL data definition file (EVD file) based on QVD
- Section 4.5 [qvd-header], page 15, – get information from QVD header, like field names as JSON, or number of records

1.2 QVD Utils in combination with other Microservices

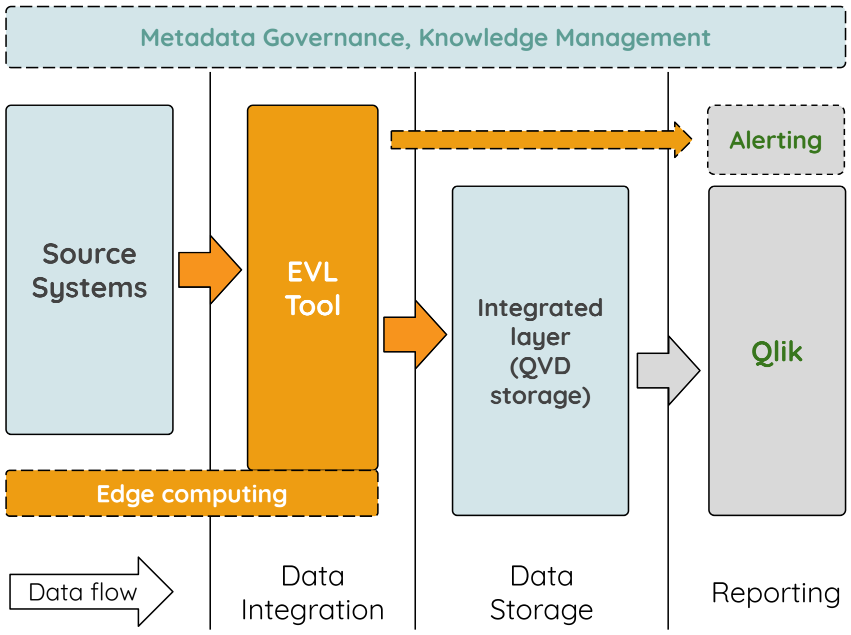


In join with other EVL Microservices, like *Hadoop Utils* or *EVL Data Hub*, you can also read/write:

- various file formats: **Avro**, **json**, **Parquet**, **xlsx**, and **xml**,
- database tables: **MySQL**, **PostgreSQL**, **Oracle**, **SQLite**, **Teradata**, or any other by ODBC.

1.3 QVD Utils with EVL Tool

Together with **EVL Tool** (an ETL tool) it can be used for example in QVD-based architecture:



2 Release Notes

Versions numbering: EVL – QVD Utils *x.y.z*

Version numbers synchronized with EVL Tool.

x – major release, i.e. big changes must happen to advance this number

y – minor releases, i.e. introduce new features

z – bugfixes

Overview

Version 0.1 (2019/10)
: Initial version of QVD Utils Microservice.
New features: write QVD files.
New utils: `csv2evd`, `csv2qvd`.

Version 0.2 (2020/04)
: Reading QVD files added.
New features: read QVD.
New utils: `qvd-header`.

Version 2.4 (2020/10)
: Version numbers synchronized with EVL Tool.
New utils: `qvd2csv`, `qvd2evd`.

Version 2.5 (2021/04)
: New options added to `qvd-header` and `qvd2csv`: `--all-as-string` and `--real-as-decimal`.

Version 2.6 (2021/10)
: New options added to `qvd2csv`: `--filter`, `--first-record`, `--guess-uniform-symbol-size`, and `--low-memory`.

Version 2.7 (2022/04)
: Version for Windows available, with simple GUI.

3 Installation and Settings

Installation

- Section 3.1 [Linux RPM], page 4, – installation on RedHat-like systems,
- Section 3.2 [Linux DEB], page 4, – installation on Debian-like systems,
- Section 3.3 [Windows], page 4, – installation on Windows 10.

Standard, Premium and Enterprise Installation

- Section 3.4 [Standard-Premium-Enterprise Installation], page 5,

3.1 Trial – Linux RPM

I.e. RedHat, CentOS, Fedora, Oracle Linux.

For *CentOS 8* firstly install required packages from *powertools* repo:

```
sudo dnf -y install dnf-plugins-core
sudo dnf config-manager --set-enabled powertools
sudo dnf install --enablerepo=powertools snappy-devel
```

Get the package for your OS from `https://www.evltool.com/downloads` and

```
tar xvf evl-qvd-utils-trial.*.rpm.tar
sudo dnf install ./evl-utils-2.8*.noarch.rpm
sudo dnf install ./evl-tool-2.8*.x86_64.rpm
sudo dnf install ./evl-qvd-utils-2.8*.x86_64.rpm
```

Then to initiate the installation for current user by

```
/opt/evl/bin/evl --init
```

3.2 Trial – Linux DEB

I.e. Ubuntu, Debian, etc.

Get the package from `https://www.evltool.com/downloads` and

```
tar xvf evl-qvd-utils-trial.ubuntu.*.deb.tar
sudo apt install ./evl-utils_2.8*_all.deb
sudo apt install ./evl-tool_2.8*_amd64.deb
sudo apt install ./evl-qvd-utils_2.8*_amd64.deb
```

Then initiate the installation for current user by

```
/opt/evl/bin/evl --init
```

3.3 Trial – Windows

Get the zip package from `https://www.evltool.com/downloads` and unzip it into some folder.

For simple graphical user interface run `qvd-utils.ps1` in PowerShell.

There are also PowerShell scripts `csv2qvd.ps1` and `qvd2csv.ps1` to run conversions programatically from command line.

3.4 Standard, Premium and Enterprise Installation

After purchase appropriate version, a key file is available at https://www.evltool.com/downloads. Just place this key file:

- on Linux systems as

  ```
  /opt/evl/etc/evl_license_key
  ```

- on Windows system just (re)place the key file as

  ```
  QVD-Utils-Windows/libexec/evl_license_key
  ```

  in the folder where the zip with QVD Utils was unzipped.

There is also possibility to set up RPM repository to get the latest version by regular system updates:

```
sudo dnf update
```

just ask support@evltool.com for details.

4 Utils

4.1 csv2evd *(since EVL 0.1)*

Read `<file.csv>` or standard input, and guess:

- data types,
- field separator (unless option '`--separator=<char>`' is used),
- if strings are quoted (unless option '`--quote=<char>`' or '`--optional-quote=<char>`' is used),
- end-of-line character(s) (unless option '`--dos-eol`' or '`--lin-eol`' or '`--mac-eol`' is used)

and write EVD to standard output or to `<file.evd>`.

It uses header line for field names, spaces are replaced by underscores.

Separator is trying to be guessed in this order: '`,`' (comma), '`;`' (semi-colon), '`|`' (pipe), '`\t`' (tab), '`:`' (colon), '' (space).

Quotation character is guessed in this order: double quotes, single quotes.

EVD is EVL data definition file, for details see man 5 evd.

Synopsis

```
csv2evd
  [<file.csv>] [-o|--output=<file.evd>]
  [--inline]
  [-d|--date=<format>]
  [-h|--header=<field_name>,...]
  [-n|--no-header]
  [-l|--null=<string>]
  [-q|--quote=<char> | --optional-quote=<char>]
  [-s|--separator=<char>]
  [-t|--datetime=<format>]
  [--timestamp=<format>]
  [--dos-eol | --lin-eol | --mac-eol]
  [-v|--verbose]

csv2evd
  ( --help | --usage | --version )
```

Options

`-d, --date=<format>`
: by default it tries only '`%Y-%m-%d`', then '`%d.%m.%Y`'

`-h, --header=<field_name>,...`
: use comma separated list of field names instead of header line, for example when there is no header in csv file (option '`-n`' must be used) or when other field names would be used

`--inline`
: output EVD in the inline format (for example to use EVD by other component with '`-d`' option)

`--dos-eol`
: do not guess end-of-line character(s), but suppose the input is text with CRLF as end of line,

`--lin-eol`
: do not guess end-of-line character(s), but suppose the input is text with LF as end of line

`--mac-eol`
: do not guess end-of-line character(s), but suppose the input is text with CR as end of line

`-n, --no-header`
: with this option it suppose there is no header. Fields will be named '`field_001`', '`field_002`', etc.

`-l, --null=<string>`
: to specify what string is used for NULL values in CSV, empty string is allowed

`-o, --output=<file.evd>`
: write output into file <file.evd> instead of standard output

`--optional-quote=<char>`
: suppose optional quote character `<char>`, must be used together with '`--separator`'

`-q, --quote=<char>`
: do not guess if fields are quoted, but suppose `<char>` as quotation character

`-s, --separator=<char>`
: do not guess the separator, but use <char> instead

`-t, --datetime=<format>`
: by default it tries only '`%Y-%m-%d %H:%M:%S`'

`--timestamp=<format>`
: by default it tries only '`%Y-%m-%d %H:%M:%S.%E*f`'

`-v, --verbose`
: print to STDERR info/debug messages

`--help`
: print this help and exit

`--usage`
: print short usage information and exit

`--version`
: print version and exit

Examples

1. Having table.csv:

   ```
   id;started;value
   1;2019-06-06;some string
   ```

   This command:

   ```
   csv2evd table.csv
   ```

will try to guess data types, field separator and if strings are quoted or not, and use header line for field names, to produce EVD to standard output:

```
id       int                null="" sep=";"
started  date("%Y-%m-%d") null="" sep=";"
value    string             null="" sep="\n"
```

2. Just an alternative invocation forwording output EVD to a file:

```
csv2evd < table.csv > table.evd
```

3. To skip header and use different field names:

```
csv2evd --header="first_field,other_field,last_one" \
  table.csv > table.evd
```

4. Case when there is no header in CSV file, but use specified field names:

```
csv2evd --no-header --header="first_field,other_field,last_one" \
  table.csv > table.evd
```

5. No header in CSV and use generated field names '`field_001`', '`field_002`', etc.:

```
csv2evd --no-header table.csv > table.evd
```

6. Consider specific date format, here day of year ('`001..366`'), and '`|`' as a field separator:

```
csv2evd --date="%j" -s '|' table.csv > table.evd
```

4.2 csv2qvd *(since EVL 0.1)*

Read `<file.csv>` with sturcture defined either in `<evd>` file or by `<inline_evd>` or guess

- data types,
- field separator (unless option '`--separator=<char>`' is used),
- if strings are quoted (unless option '`--quote=<char>`' or '`--optional-quote=<char>`' is used),
- end-of-line character(s) (unless option '`--dos-eol`' or '`--lin-eol`' or '`--mac-eol`' is used)

and write QVD file to `<file.qvd>` or standard output. For guessing data types (EVD) it uses utility '`csv2evd`'.

EVD is EVL data definition file, for details see man 5 evd.

Synopsis

```
csv2qvd
  <file.csv>
  [-o|--output=<file.qvd>]
  [-d|--date=<format>]
  [-h|--header=<field_name>,...]
  [-n|--no-header]
  [-l|--null=<string>]
  [-q|--quote=<char> | --optional-quote=<char>]
  [-s|--separator=<char>]
  [-t|--datetime=<format>]
  [--timestamp=<format>]
  [--dos-eol | --lin-eol |--mac-eol]
  [-v|--verbose]

csv2qvd
  <file.csv> (<evd>|-d <inline_evd>)
```

```
    [-o|--output=<file.qvd>]
    [--dos-eol | --lin-eol |--mac-eol]
    [-v|--verbose]

  csv2qvd
    ( --help | --usage | --version )
```

Options

Standard options:

`-d, --data-definition=<inline_evd>`
: either this option or the file `<evd>` must be presented to use already defined EVD

`--dos-eol`
: do not guess end-of-line character(s), but suppose the input is text with CRLF as end of line,

`--lin-eol`
: do not guess end-of-line character(s), but suppose the input is text with LF as end of line,

`--mac-eol`
: do not guess end-of-line character(s), but suppose the input is text with CR as end of line,

`-o, --output=<file.qvd>`
: write output into `<file.qvd>` instead of standard output

`-v, --verbose`
: print to STDERR info/debug messages

`--help`
: print this help and exit

`--usage`
: print short usage information and exit

`--version`
: print version and exit

EVD options:

`--date=<format>`
: by default it tries only '`%Y-%m-%d`', then '`%d.%m.%Y`'

`-h, --header=<field_name>,...`
: use comma separated list of field names instead of header line, for example when there is no header in csv file (option '`-n`' must be used) or when other field names should be used

`-n, --no-header`
: with this option it suppose there is no header. Fields will be named '`field_001`', '`field_002`', etc.

`-l, --null=<string>`
: to specify what string is used for NULL values in CSV, empty string is allowed

`--optional-quote=<char>`
: suppose optional quote character `<char>`, must be used together with '`--separator`'

`-q, --quote=<char>`
: do not guess if fields are quoted, but suppose `<char>` as quotation character

`-s, --separator=<char>`
: do not guess the separator, but use `<char>` instead

`-t, --datetime=<format>`
: by default it tries only '`%Y-%m-%d %H:%M:%S`'

`--timestamp=<format>`
: by default it tries only '`%Y-%m-%d %H:%M:%S.%E*f`'

Examples

1. Having '`some.csv`':
   ```
   id;started;value
   1;2019-06-06;some string
   ```
   The command:
   ```
   csv2qvd --null="NULL" some.csv > some.qvd
   ```
   will produce some.qvd file with these field:
   ```
   id       int                null="NULL"  sep=";"
   started  date("%Y-%m-%d")   null="NULL"  sep=";"
   value    string             null="NULL"  sep="\n"
   ```
2. Following invocation will guess data types, field separator and if strings are quoted or not, and use header line for field names:
   ```
   csv2qvd table.csv > table.qvd
   ```
   With the '`--verbose`' option it will write to standard error the whole EVD file which was used:
   ```
   csv2qvd --verbose table.csv > table.qvd
   ```
3. To skip header and use different field names:
   ```
   csv2qvd --header="first_field,other_field,last_one"
     table.csv > table.qvd
   ```
4. Case when there is no header in CSV file, but use specified field names:
   ```
   csv2qvd --no-header --header="first_field,other_field,last_one" \
     table.csv > table.qvd
   ```
5. No header in CSV and use generated field names '`field_001`', '`field_002`', etc.:
   ```
   csv2qvd --no-header table.csv > table.qvd
   ```
6. Consider specific date format, here day of year ('`001..366`'), and '`|`' as a field separator:
   ```
   csv2qvd --date="%j" -s '|' table.csv > table.qvd
   ```
7. To use own (specific or already generated) EVD file (i.e. data types definition):
   ```
   csv2qvd table.csv table.evd > table.qvd
   ```

Another example

Let's have following `sample_1.csv` file with header and with semicolon as a field separator:

```
ID;Name;Code;Price;Created at
1;item_11;11;15,85;2016-05-04
2;item_12;12;21,25;2019-11-11
3;item_13;13;12,99;2019-05-05
```

Then by running:

```
csv2qvd -v < sample_1.csv > sample_1.qvd
```

will guess data types and use field names from the header and produce `sample_1.qvd` file. And it will also write to standard error EVL data types which were used:

```
Used EVD:
---------
ID         int           null="" sep=";"
Name       string        null="" sep=";"
Code       int           null="" sep=";"
Price      decimal(8,2)  null="" sep=";"
Created_at date          null="" sep="\n"
---------
```

More complex example

If guessed data types are not correct for some reason, for example the date format would be different or field which looks like an integer might be next time a string, then it is better to provide own EVD file with EVL data definition description.

Suppose following `sample_2.csv` file with header and with semicolon as a field separator:

```
ID;Name;Code;Price;Created at
1;item_11;11;15,85;05/04/2016
2;item_12;12;21,25;11/11/2019
3;item_13;13;12,99;05/05/2019
```

Let's start by guessing EVD by `csv2evd` utility:

```
csv2evd sample_2.csv > sample_2.evd
```

It will result in `sample_2.evd`:

```
ID         int          null="" sep=";"
Name       string       null="" sep=";"
Code       int          null="" sep=";"
Price      decimal(8,2) null="" sep=";"
Created_at string       null="" sep="\n"
```

We can either use option `--date="%m/%d/%Y"` or simply edit `sample_2.evd` manually. Both will result in:

```
ID         int                null="" sep=";"
Name       string             null="" sep=";"
Code       int                null="" sep=";"
Price      decimal(8,2)       null="" sep=";"
Created_at date("%m/%d/%Y") null="" sep="\n"
```

4.3 qvd2csv *(since EVL 2.4)*

Read `<file.qvd>` and write CSV file to `<file.csv>` or standard output. It uses data types from QVD header or from existing `<evd>` file or from `<inline_evd>`.

EVD is EVL data definition file, for details see man 5 evd.

Synopsis

```
qvd2csv
  <file.qvd>
  [-o|--output=<file.csv>]
  [--all-as-string | --real-as-decimal[=<precision>,<scale>]]
  [-d|--date=<format>]
  [-h|--header=<field_name>,...]
```

```
    [-n|--no-header]
    [-l|--null=<string>]
    [-q|--quote=<char>]
    [-s|--separator=<char>]
    [-t|--datetime=<format>]
    [-a|--dos-eol | -b|--mac-eol]
    [--filter=<condition>]
    [--first-record=<n>]
    [--guess-uniform-symbol-size]
    [--low-memory]
    [-v|--verbose]

  qvd2csv
    <file.qvd> (<evd>|-d <inline_evd>)
    [-m|--match-fields]
    [-o|--output=<file.csv>]
    [-h|--header=<field_name>,...]
    [-n|--no-header]
    [-a|--dos-eol | -b|--mac-eol]
    [--filter=<condition>]
    [--first-record=<n>]
    [--guess-uniform-symbol-size]
    [--low-memory]
    [-v|--verbose]

  qvd2csv
    ( --help | --usage | --version )
```

Options

`--all-as-string`
: interpret all fields as strings. (Since EVL 2.5.)

`-d, --data-definition=<inline_evd>`
: either this option or the file `<evd>` must be presented to use already defined (custom) EVD

`-a, --dos-eol`
: output DOS end-of-line, i.e. CR+LF (‘`\r\n`’)

`-b, --mac-eol`
: output Mac end-of-line, i.e. CR (‘`\r`’)

`--date=<format>`
: to specify a `<format>` for date data type

`--filter=<condition>`
: read only records with given `<condition>`. (Since EVL 2.6.)

`--first-record=<n>`
: start to read from the record number `<n>`. (Since EVL 2.6.)

`--guess-uniform-symbol-size`
: might speed up indexing of dictionary, but it could not work in all cases. Use only in special cases when need really good performance. (Since EVL 2.6.)

`-h, --header=<field_name>,...`
: use comma separated list of field names instead of header line, for example when you don't want to use field names from QVD header.

`--low-memory`
: do not read dictionary into memory. This could save memory consumption, but slows down reading the source file. (Since EVL 2.6.)

`-l, --null=<string>`
: to specify what string is used for NULL values in CSV, empty string is allowed

`-m, --match-fields`
: to read only a subset of fields from QVD file or to read them in different order

`-n, --no-header`
: with this option it produces no header line

`-o, --output=<file.csv>`
: write output into `<file.csv>` instead of standard output

`-q, --quote=<char>`
: to use quoted fields for the CSV output. When data contains such `<char>`, all of them are escaped by duplicating them. For example using '`--quote="\""`' will serve data like '`some "text"`' as '`"some ""text"""`'.

`--real-as-decimal[=<precision>,<scale>]`
: interpret '`real`' data types as '`decimal(<precision>,<scale>)`'. When no `<precision>` or `<scale>` is specified, use values from environment variables '`EVL_DEFAULT_DECIMAL_PRECISION`' and '`EVL_DEFAULT_DECIMAL_SCALE`', which are by default set to 18 and 2. (Since EVL 2.5.)

`-s, --separator=<char>`
: to use `<char>` as field separator for the CSV output

`-t, --datetime=<format>`
: to specify a `<format>` for datetime data type

`-v, --verbose`
: print to standard error output info/debug messages

`--help`
: print this help and exit

`--usage`
: print short usage information and exit

`--version`
: print version and exit

Examples

1. Having '`some.qvd`', the command to produce CSV file with empty strings representing NULL values, dates in format '`DD.MM.YYYY`' and with Windows end-of-line (i.e. CRLF):

   ```
   qvd2csv --null="" --date="%d.%m.%Y" --dos-eol some.qvd > some.csv
   ```

2. To filter only particular records from '`large.qvd`', for example we would like to read only latest records represented by field '`invoice_id`':

   ```
   qvd2csv --filter="invoice_id>7654000" large.qvd > latest.csv
   ```

3. To cut only particular columns from '`large.qvd`', for example only column '`invoice_id`':

   ```
   qvd2csv --match-fields -d 'invoice_id int null=""' large.qvd > latest.csv
   ```

4. To read only after by some number of rows:
   ```
   qvd2csv --first-record=1234000 huge.qvd > latest.csv
   ```
   This could be quite useful when reading a huge QVD file.

4.4 qvd2evd *(since EVL 2.4)*

Read header of `<file.qvd>` or standard input, guess data types, and write EVD to standard output or to `<file.evd>`.

EVD is EVL data definition file, for details see man 5 evd.

Synopsis

```
qvd2evd
  [<file.qvd>] [-o|--output=<file.evd>]
  [--all-as-string | --real-as-decimal[=<precision>,<scale>]]
  [-d|--date=<format>]
  [--inline]
  [-l|--null[=<string>]]
  [-q|--quote=<char>]
  [-r|--record-separator=<char>]
  [-s|--field-separator=<char>]
  [-t|--datetime=<format>]
  [-v|--verbose]

qvd2evd
  ( --help | --usage | --version )
```

Options

`--all-as-string`
: produce EVD with all fields as strings. (Since EVL 2.5.)

`-d, --date=<date_format>`
: use format argument for date data type

`--inline`
: output EVD in the inline format (for example to use EVD by other component with '`-d`' option)

`-l, --null=<string>`
: to specify what string is used for NULL values in QVD, empty string is allowed

`-o, --output=<file.evd>`
: write output into file <file.evd> instead of standard output

`-q, --quote=<char>`
: to use a quote argument in EVD

`--real-as-decimal[=<precision>,<scale>]`
: produce EVD with '`decimal(<precision>,<scale>)`' instead of '`double`'. When no `<precision>` or `<scale>` is specified, it uses values from environment variables '`EVL_DEFAULT_DECIMAL_PRECISION`' and '`EVL_DEFAULT_DECIMAL_SCALE`', which are by default set to 18 and 2. (Since EVL 2.5.)

`-r, --record-separator=<char>`
: use '`sep="<char>"`' for last field

`-s, --field-separator=<char>`
: add ‘`sep="<char>"`’ to each field, except the last one

`-t, --datetime=<format>`
: use format for datetime data type by default it produces no format for datetime, so the EVL_DEFAULT_DATETIME_PATTERN is then used (which is by default set to ‘`"%Y-%m-%d %H:%M:%S"`’)

`-v, --verbose`
: print to STDERR info/debug messages

`--help`
: print this help and exit

`--usage`
: print short usage information and exit

`--version`
: print version and exit

Examples

1. Having ‘`some.qvd`’, this command:

   ```
   qvd2evd --null -r '\n' -s ';' -d '%d.%m.%Y' some.qvd
   ```

   will produce:

   ```
   id       int                null="" sep=";"
   started  date("%d.%m.%Y") null="" sep=";"
   value    string             null="" sep="\n"
   ```

4.5 qvd-header *(since EVL 0.2)*

Take the header of `<file.qvd>` or standard input and produce to standard output particular information, for example EVL data definition file or number of records.

Synopsis

```
qvd-header
  [<file.qvd>] --output=evd
  [--all-as-string | --real-as-decimal[=<precision>,<scale>]]
  [-d|--date=<format>]
  [--inline]
  [-l|--null=<string>]
  [-q|--quote=<char>]
  [-r|--record-separator=<char>]
  [-s|--field-separator=<char>]
  [-t|--datetime=<format>]
  [-v|--verbose]

qvd-header
  [<file.qvd>] --output=(json|xml)
  [--fields]
  [-v|--verbose]

qvd-header
  [<file.qvd>]
```

```
       [ --table-name | --no-of-records | --fields | --tag=<xml_tag_name> ]
       [-v|--verbose]

     qvd-header
       ( --help | --usage | --version )
```

Options

`--no-of-records`
: return the value of '`NoOfRecords`' tag

`--fields`
: provide only fields' information

`--table-name`
: return the value of '`TableName`'

`--tag=<xml_tag_name>`
: return the value of `<xml_tag_name>`

Output Options:

`--output=evd`
: return EVD data types definition

`--output=json`
: return information as JSON

`--output=xml`
: return information as XML

EVD options:

`--all-as-string`
: produce EVD with all fields as strings. (Since EVL 2.5.)

`-d, --date=<date_format>`
: use format argument for date data type

`--inline`
: output EVD, XML or JSON in the inline format (for example to use EVD by other component with '`-d`' option)

`-l, --null=<string>`
: to specify what string is used for NULL values in EVD

`-q, --quote=<char>`
: to use a quote argument in EVD

`--real-as-decimal[=<precision>,<scale>]`
: produce EVD with '`decimal(<precision>,<scale>)`' instead of '`double`'. When no `<precision>` or `<scale>` is specified, it uses values from environment variables '`EVL_DEFAULT_DECIMAL_PRECISION`' and '`EVL_DEFAULT_DECIMAL_SCALE`', which are by default set to 18 and 2. (Since EVL 2.5.)

`-r, --record-separator=<char>`
: use '`sep="<char>"`' for last field

`-s, --field-separator=<char>`
: use '`sep="<char>"`' for each field

`-t, --datetime=<date_format>`
: use format for datetime data type

Standard options:

`--help`

print this help and exit

`--usage`

print short usage information and exit

`-v, --verbose`

print to stderr info/debug messages of the component

`--version`

print version and exit

Examples

1. EVD example:

   ```
   qvd-header some.qvd --output=evd --record-separator="\n" \
                        --null="" --date="%Y-%m-%d"
   ```

   will produce for example:

   ```
   id          int               null=""
   some_stamp  datetime          null=""
   some_date   date("%Y-%m-%d") null=""
   value       string            null="" sep="\n"
   ```

2. JSON example:

   ```
   qvd-header some.qvd --output=json --fields
   ```

   will produce for example:

   ```
   {
     "fields":
     [
       {
         name: "REQUEST_HOUR",
         type: "timestamp",
         format: "%Y-%m-%d"
       },
       ...
     ]
   }
   ```

   And:

   ```
   qvd-header some.qvd --output=json
   ```

   will produce for example:

   ```
   {
     "name": "Table1",
     "records": 3615,
     "fields":
     [
       {
         name: "REQUEST_HOUR",
         type: "timestamp",
         format: "%Y-%m-%d",
         "tags": [
           "$numeric",
   ```

```
        "$timestamp"
      ]
    },
    ...
  ]
}
```

Appendix A EVD and Data Types

'`EVD`' stands for '`EVL Data Definition`' and it is the way how to specify structure of data sets in EVL. It can be used either inline, as a component option, or in an `*.evd` file.

EVL uses mostly standard C++ data types, so most of them are well known.

A.1 EVD Structure

Example first: Let's have a CSV file:

```
1;Otto Wichterle;27.10.1913;12,345.78;2025-03-19 14:34:07
2;;1.1.1970;0.00;2025-03-19 14:35:44
```

then following `evd` file would describe its structure:[1]

```
ID           int                 sep=";"
Name         string              sep=";"  null=""
"Birth Date" date("%-d.%-m.%Y") sep=";"  null="1.1.1970"
Amount       decimal(12,3)       sep=";"    thousands_sep=","
"Created At" datetime            sep="\n" null="0000-00-00 00:00:00"
```

In general each nonempty line of EVD file looks like this:

```
<indent> Field_Name <blank> Data_Type <blank> EVD_Options
```

where

'`<indent>`'

might be empty, 2 spaces, 4 spaces, 6 spaces, etc., to define a substructure of compound data types, see Section A.8 [Compound Types], page 28, for details.

'`Field_Name`'

is a sequence of any printable ASCII characters below 128. When a space is used, then whole field name must be quoted by double quotes. Special characters (also only ASCII ones under 128) must be escaped, e.g. '`\n`', '`\r`', '`\t`', '`\v`', '`\b`', '`\f`', '`\a`', '`\"`', '`\\`', or in hexa '`\x??`'. Characters other than letters, numbers and underscore are replaced by underscore in mappings. All these field names are valid:

```
                            // Name in mapping:
recommended_field_name      // recommended_field_name
"Field with a Space"        // Field_with_a_Space
'field-with-a-hyphen'       // _field_with_a_hyphen_
"$field_with_dollar"        // _field_with_dollar
'single_quoted'field'       // _single_quoted_field_
"with\nnewline"             // with_newline
```

'`Data_Type`'

is one of:

- `string`, `ustring`, see Section A.3 [String], page 22,
- `char`, `uchar`, `short`, `ushort`, `int`, `uint`, `long`, `ulong`, `int128`, `uint128`, see Section A.4 [Integral Types], page 23,
- `decimal`, see Section A.5 [Decimal], page 23,

[1] By setting environment variables `EVL_DEFAULT_FIELD_SEPARATOR=";"` and `EVL_DEFAULT_RECORD_SEPARATOR=$'\n'` one can avoid to use `sep` options.

- `float`, `double`, see Section A.6 [Float and Double], page 24,
- `date`, `time`, `time_ns`, `interval`, `interval_ns`, `datetime`, `timestamp`, see Section A.7 [Date and Time], page 24,

'`EVD_Options`'
: is `<blank>` separated list of options, see Section A.2 [EVD Options], page 20,

'`<blank>`'
: is one or more spaces and/or tabs.

A.1.1 Comments

Standard C-style comments can be used in `evd` file, for example:

```
street_id    int
street_name  string
street_code  string  null=""  // but NOT NULL in DB
/* COMBAK: street_code will be replaced by street_num later this year
street_num   long
*/
```

A.1.2 Inline EVD

For the most of the EVL Components an inline EVD can be specified as an option. In such case comments are not allowed and the format is simply the same as for EVD in a file, just instead of newlines, commas are used to separate each field definition.

The same structure, as in above EVD Example, but as a component option (a comma separated list of fields with data types and options):

```
--data-definition='id int sep=";",
    name string sep=";" null="",
    birth_date date sep=";" null="1970-01-01",
    amount decimal(12,3) sep=";" thousands_sep=",",
    created_at datetime sep="\n" null="0000-00-00 00:00:00"'
```

A.2 EVD Options

Structure of the data is described in an EVD file – an EVL data types definition file – with file extension `.evd`.

A.2.1 Separator Definition

Field separator is defined by '`sep="X"`', where '`X`' can be an empty string or an ascii character below 128 specified as normal string or special character '`\n`', '`\r`', '`\t`', '`\v`', '`\b`', '`\f`', '`\a`', '`\"`', '`\\`', or in hexa '`\x??`' (0-7E) (as it is always a single character, '`\x?`' is also possible).

Default separators can be defined:

`EVL_DEFAULT_FIELD_SEPARATOR`
: defines default field separator, when not set, `EVL_DEFAULT_FIELD_SEPARATOR='|'` is used,

`EVL_DEFAULT_RECORD_SEPARATOR`
: defines default record separator, i.e. the last field separator, when not set, `EVL_DEFAULT_RECORD_SEPARATOR='\n'` is used.

When these variables are set, then no '`sep=`' options are needed in the above `EVD` example and these defaults are used instead.

> **Note:** It is recommended to use these variables only for project-wide settings in `project.sh`. Try to avoid to set them in jobs. Better use '`sep=`' option in `evd` file.

In case we want to have an empty separator, for example after fixed length field, we can use `sep=""`.

A.2.2 Null Option

A null string by `null="X"` or list of strings `null=["X","Y",...]` can be specified. Then such string(s) will be read as `null` values when `--text-input` is used by the component.

When writing the `null` value by the output component with `--text-output` option, such string will be used instead.

When the list of null values is specified, then the first one will be used to write.

To type a special character, like newline or `TAB`, standard hexadecimal notation can be used: `\x??`, or also special notation for often used special characters: `\n`, `\r`, `\t`, `\v`, `\b`, `\f`, `\a`, `\"`, `\\`. So then to interpret a tabulator as NULL value use `null="\t"`.

A.2.3 Quote Option

When reading `csv` files, fields might be quoted by some character, usually by double quotes: `"`.

Proper parsing of such field is done by specifying attributes `quote=` or `optional_quote=`.

Specified string might be any ascii character below 128 specified as normal string or special character `\n`, `\r`, `\t`, `\v`, `\b`, `\f`, `\a`, `\"`, `\\`, or in hexa `\x??` (0-7E) (as it is always a single character, `\x?` is also possible).

`quote="<quote_char>"`
: Use this attribute when the field is always quoted.

`optional_quote="<quote_char>"`
: Using this attribute, the field doesn't need to be quoted.

A.2.4 Encoding and Locale *(since EVL 2.5)*

`enc="<encoding>"`
: To specify an encoding of given field, string functions then behaves according to that.

`locale="<locale>"`
: To specify a locale of given field, components (like sort) then behaves according to that.

Examples

```
czech_string_in_utf8  string  enc="utf8" locale="cs_CZ"
en_string_in_utf8     string  enc="utf8" locale="en_GB"
```

When there is no encoding or locale specified in an EVD, then following environment variables can be used:

`EVL_DEFAULT_STRING_ENC=""`
: defines default encoding, when not set, empty encoding is used,

`EVL_DEFAULT_STRING_LOCALE="C"`
: defines default locale, when not set, generic `C` locale is used.

A.2.5 Max string length *(since EVL 2.5)*

Attributes which are used to specify maximal length of given string field. So far used only in case of load/unload tables.

`max_bytes="<number>"`
: To specify maximum Bytes of given string field. Is populated when generated based on table definition, e.g. `VARCHAR(100 BYTES)`.

`max_chars="<number>"`
: To specify maximum characters of given string field. Is populated when generated based on table definition, e.g. '`VARCHAR(100 CHARS)`'.

Examples:

```
string_20_bytes  string  enc="utf8" max_bytes="20"  // VARCHAR(20 BYTES)
string_20_chars  string  enc="utf8" max_chars="20"  // VARCHAR(20 CHARS)
```

Both attributes are currently used in '`Writeora`' component to know the maximal length of a string field.

A.2.6 QVD options *(since EVL 2.4)*

`qvd:format="<format_string>"`
: To specify a format string for '`timestamp`', '`datetime`', and '`date`' data types when read/write Qlik's QVD files. Example:

```
request_dt  timestamp   qvd:format="%d/%m/%Y %H:%M:%S"
some_date   date        qvd:format="%d.%m.%Y"
```

`qvd:interval`
`qvd:time` To be used as an attribute for '`timestamp`' and '`datetime`' data types to get an interval or time data type in Qlik's QVD files. Example:

```
request_time1  timestamp  qvd:time
request_time2  timestamp  qvd:interval
```

Compared to Qlik's time data type, interval can be larger than 24 hours. For example input timestamp '`1970-01-02 03:05:30`' would be '`03:05:30`' as time, but '`27:05:30`' as interval.

A.3 String

Standard C++ library '`std::basic_string`' is used for strings. For details see

http://en.cppreference.com/w/cpp/string/basic_string

`string` size: up to 2^{64} Bytes (i.e. limited only by memory)

An EVD file Example:

```
field_name1  string(10)
field_name2  string(10) sep=""
field_name3  string     sep=";" null="NULL"
field_name4  string             null=""              quote="\""
field_name5  string             null=["","N/A","NA"]
last_field   string
```

where

'`field_name1`'
: cannot be NULL and has fixed length 10 bytes, followed by the value of `$EVL_DEFAULT_FIELD_SEPARATOR` environment variable.

'`field_name2`'
: cannot be NULL and has fixed length 10 bytes, with no separator.

'`field_name3`'
: is nullable and string '`NULL`' is interpreted as NULL value. End of the field is represented by character '`;`'.

'`field_name4`'
: is nullable and empty string is interpreted as NULL value. Field is quoted by '`"`', but for an empty string, quotes are not needed. The end of the field is represented by `$EVL_DEFAULT_FIELD_SEPARATOR`.

'`field_name5`'
: is nullable and empty string, '`N/A`' and '`NA`' are interpreted as NULL value when reading, but when writing into text file, NULL is represented by the first one, i.e. an empty string. The end of the field is represented by `$EVL_DEFAULT_FIELD_SEPARATOR`.

'`last_field`'
: cannot be NULL and the end of the field is represented by `$EVL_DEFAULT_RECORD_SEPARATOR`.

Example of four records which can be parsed by above EVD file definition.

```
          |          NULL;"second string field"|NA|last field
0123456789|0123456789first string field;""|N/A|last field
----------|----------;"  ;  second field  |  "|third string field|last field
abcdefghij|abcdefghij       ;||last field
```

Neither `EVL_DEFAULT_FIELD_SEPARATOR` nor `EVL_DEFAULT_RECORD_SEPARATOR` is set, so default values are used, i.e. '`|`' and '`\n`'.

A.4 Integral Types

All integral data types are standard C++ ones.

`char`	size: 1 Byte, min: -128, max: 127
`uchar`	size: 1 Byte, min: 0, max: 255
`short`	size: 2 Bytes, min: $-32\,768$, max: 32 767
`ushort`	size: 2 Bytes, min: 0, max: 65 535
`int`	size: 4 Bytes, min: $-2\,147\,483\,648$, max: 2 147 483 647
`uint`	size: 4 Bytes, min: 0, max: 4 294 967 295
`long`	size: 8 Bytes, min: -2^{63} (approx. -9×10^{18}), max: $2^{63}-1$ (approx. 9×10^{18})
`ulong`	size: 8 Bytes, min: 0, max: $2^{64}-1$ (approx. 18×10^{18})
`int128`	size: 16 Bytes, min: -2^{127} (approx. -1.7×10^{38}), max: $2^{127}-1$ (approx. 1.7×10^{38})
`uint128`	size: 16 Bytes, min: 0, max: $2^{128}-1$ (approx. 3.4×10^{38})

Except '`sep=`', '`null=`', '`quote=`', '`optional_quote=`', no other options are possible for these data types.

A.5 Decimal

Decimal data type is defined by '`decimal(m,n)`', where '`m`' is number of all digits and '`n`' is the number of decimal places. Decimal is EVL custom data type.

`decimal(m,n)`
: when '`n`' is missing, zero is supposed
size: 8 Bytes for '`m`' up to 18 digits
size: 16 Bytes for '`m`' from 19 to 38 digits

Next to standard EVD options (i.e. ‘`sep=`’, ‘`null=`’, ‘`quote=`’, ‘`optional_quote=`’) decimal and thousands separator can be specified:

`decimal_sep="."`
: to specify a decimal separator, which can be any single ascii character below 128; by default it is a decimal point

`thousands_sep=""`
: defines how to separate thousands, it can be any single ascii character below 128; by default there is no thousands separator.

An EVD file example:

```
revenues  decimal(9,4)  decimal_sep="," thousands_sep="."  // e.g. 12.345,6789
expenses  decimal(18)                                 // e.g. 123456789012345678
taxes     decimal(18,6) thousands_sep=" "             // e.g. 123 456 789 012.345678
latitude  decimal(10,6)                               // e.g. 49.8197203
longitude decimal(10,6) decimal_sep=","               // e.g. 18,1673552
```

A.6 Float and Double

Float and double are standard C++ data types.

`float` size: 4 Bytes, range: $\pm$ 3.4 $\times 10^{\pm 38}$ (about 7 digits)

`double` size: 8 Bytes, range: $\pm$ 1.7 $\times 10^{\pm 308}$ (about 15 digits)

Except ‘`sep=`’, ‘`null=`’, ‘`quote=`’, ‘`optional_quote=`’, no other options are possible for these data types.

Note: Compared to `decimal(m,n)` data type, operating with floats and doubles (doing summations for example), usually leads to approximated values. So it is usually good idea to avoid using these data types for money and such.

Example

With EVD file

```
sent_mb      float  sep="|"  null=""
received_mb  float  sep="\n" null=""
```

you can read source `csv` file like this:

```
0.321e12|1.234E-02
12.78E11|3.798
```

A.7 Date and Time

Date, time, time_ns, interval, interval_ns and datetime are data types based on standard C++ library ‘`std::time`’. Timestamp is built upon Google’s ‘`cctz`’ library.

`date` *(since EVL 1.0)*
: to store a date, i.e. day, month and year
size: 4 Bytes, range: 1970-01-01 $\pm$ approx. 6 $\times 10^{11}$ years
first 2 Bytes keeps a year, then 1 Byte for month and 1 Byte for day
example: `2008-04-20`

`time` *(since EVL 2.8)*
: to store a day time, i.e. hour, minute and second

size: 4 Bytes, range: 00:00:00 – 23:59:59
example: `13:35:00`

`time_ns` *(since EVL 2.8)*
: to store a day time with nanoseconds
size: 8 Bytes, range: 00:00:00.000000000 – 23:59:59.999999999
example: `13:37:00.350000000`

`interval` *(since EVL 2.8)*
: to store a time interval in hours, minutes and seconds
size: 4 Bytes, min: 00:00:00
example: `165:35:00`

`interval_ns` *(since EVL 2.8)*
: to store a time interval with nanoseconds
size: 8 Bytes, min: 00:00:00.000000000
example: `165:35:00.123456789`

`datetime` *(since EVL 1.0 as timestamp, since EVL 2.4 as datetime)*
: to store a date and time, i.e. year, month, day, hour, minute and second
size: 8 Bytes, range: 1970-01-01 00:00:00 ± approx. 6×10^{11} years
example: `2010-07-01 09:02:00`

`timestamp` *(since EVL 2.4)*
: to store a date and time with nanoseconds and with a time zone, i.e. year, month, day, hour, minute, second, nanoseconds and possibly a time zone
size: 12 Bytes, range: 1970-01-01 00:00:00 ± approx. 6×10^{11} years
example: `2015-05-09 13:37:00.000 +02:00`

A.7.1 Format string

As an argument (in curly brackets) formatting pattern can be specified. Standard C notation is used.

When no argument to date and time data types are provided, defaults are used:

`EVL_DEFAULT_DATE_PATTERN`
: to specify default formatting string for '`date`' data type,
by default it is `"%Y-%m-%d"`

`EVL_DEFAULT_TIME_PATTERN`
: to specify default formatting string for '`time`' data type,
by default it is `"%H:%M:%S"`

`EVL_DEFAULT_DATETIME_PATTERN`
: to specify default formatting string for '`datetime`' data type,
by default it is `"%Y-%m-%d %H:%M:%S"`

`EVL_DEFAULT_TIMESTAMP_PATTERN`
: to specify default formatting string for '`timestamp`' data type,
by default it is `"%Y-%m-%d %H:%M:%E*S"`

All possible format strings:

`%%`	a literal '%'
`%a`	locale's abbreviated weekday name (e.g. '`Sun`')
`%A`	locale's full weekday name (e.g. '`Sunday`')
`%b`	locale's abbreviated month name (e.g. '`Jan`')

%B	locale's full month name (e.g. 'January')
%c	locale's date and time (e.g. 'Thu Mar 3 23:05:25 2005')
%C	century; like '%Y', except omit last two digits (e.g. '20')
%d	day of month (e.g. '01')
%D	date; same as '%m/%d/%y'
%e	day of month, space padded; same as '%_d'
%Ez	RFC3339-compatible numeric UTC offset (+hh:mm or -hh:mm)
%E*z	full-resolution numeric UTC offset (+hh:mm:ss or -hh:mm:ss)
%E#S	seconds with # digits of fractional precision
%E*S	seconds with full fractional precision (a literal '*')
%E#f	fractional seconds with # digits of precision
%E*f	fractional seconds with full precision (a literal '*')
%E4Y	four-character years (-999 ... -001, 0000, 0001 ... 9999)
%ET	the RFC3339 "date-time" separator "T"
%F	full date; same as '%Y-%m-%d'
%g	last two digits of year of ISO week number (see '%G')
%G	year of ISO week number (see '%V'); normally useful only with '%V'
%h	same as '%b'
%H	hour ('00'..'23')
%I	hour ('01'..'12')
%j	day of year ('001'..'366')
%k	hour, space padded (' 0'..'23'); same as '%_H'
%l	hour, space padded (' 1'..'12'); same as '%_I'
%m	month ('01'..'12')
%M	minute ('00'..'59')
%n	a newline
%p	locale's equivalent of either 'AM' or 'PM'; blank if not known
%P	like '%p', but lower case
%r	locale's 12-hour clock time (e.g. '11:11:04 PM')
%R	24-hour hour and minute; same as '%H:%M'
%s	seconds since '1970-01-01 00:00:00 UTC'
%S	second ('00'..'60')
%t	a tab
%T	time; same as '%H:%M:%S'
%u	day of week ('1'..'7'); '1' is Monday
%U	week number of year, with Sunday as first day of week ('00'..'53')

`%V`	ISO week number, with Monday as first day of week (‘`01`’..‘`53`’)
`%w`	day of week (‘`0`’..‘`6`’); ‘`0`’ is Sunday
`%W`	week number of year, with Monday as first day of week (‘`00`’..‘`53`’)
`%x`	locale’s date representation (e.g. ‘`12/31/99`’)
`%X`	locale’s time representation (e.g. ‘`23:13:48`’)
`%y`	last two digits of year (‘`00`’..‘`99`’)
`%Y`	year
`%z`	+hhmm numeric time zone (e.g., -0400)
`%Z`	alphabetic time zone abbreviation (e.g., EDT)

By default, date pads numeric fields with zeroes. The following optional flags may follow ‘`%`’.

`-`	(hyphen) do not pad the field
`_`	(underscore) pad with spaces
`0`	(zero) pad with zeros
`^`	use upper case if possible
`#`	use opposite case if possible

A.7.2 EVD Example

Following dates definition are equivalent.

```
valid_from  date
valid_from  date("%F")
valid_from  date("%Y-%m-%d")
```

Following datetimes are all the same.

```
request_dt  datetime
request_dt  datetime("%F %T")
request_dt  datetime("%Y-%m-%d %H:%M:%S")
```

Following timestamps are all the same.

```
request_dt  timestamp
request_dt  timestamp("%F %T.%E9f")
request_dt  timestamp("%Y-%m-%d %H:%M:%S.%E9f")
```

QVD’s format string can be specified:

```
request_dt  timestamp  qvd:format="%d/%m/%Y %H:%M:%S"
some_date   date       qvd:format="%d.%m.%Y"
```

A.7.3 Qlik’s time

When time need to be specified in QVD file, then standard timestamp need to be provided, just with ‘`qvd:time`’ option. Then the date is simply cut off from the timestamp to be stored in QVD:

```
request_time  timestamp("%H:%M:%S")  qvd:time
```

A.7.4 Qlik’s interval

When interval data type need to be specified in QVD file, then standard timestamp need to be provided, just with ‘`qvd:interval`’ option. Then the time is taken since ‘`1970-01-01`’:

```
request_time  timestamp("%Y-%m-%d %H:%M:%S")  qvd:interval
```

Note: Compared to Qlik’s time data type, interval can be larger than 24 hours. For example input timestamp ‘`1970-01-02 03:05:30`’ would be ‘`03:05:30`’ as time, but ‘`27:05:30`’ as interval.

A.8 Compound Types

Note: Compound data types cannot be used to read/write QVD/QVX files. These file formats do not support it.

Appendix B Software License, Support and Service Terms

Last modified: January 15, 2021

These Software License, Support and Service Terms (the **Terms**) shall apply to any use of the EVL QVD Utils, as this term is defined further below (the **Software**), unless explicitly agreed otherwise by the Parties.

The Licensee as this term is defined below hereby confirms that the Licensee read these terms carefully before downloading, installing or otherwise using the software. Download or other use of the software will be interpreted as full and unconditional acceptance of the terms and the Licensee shall be bound by all terms and conditions contained further below in these terms.

Definitions

Affiliate means a relative (in case of person) or an entity that directly or indirectly controls, is controlled by, or is under common control of the same person or group of people (in case of corporation or another legal entity).

EVL Tool, s.r.o.
means EVL Tool, s.r.o., Business ID No. 04522249, with its registered office at Litevská 1174/8, Prague 10, Post Code 10000, registered in the Commercial Register under file No. C249060 kept by the Municipal Court in Prague.

EVL Tool Indemnitees
means Licensor and its directors, officers, employees, suppliers, consultants, contractors, and agents.

Confidential Information
has the meaning set in Section [Confidential Information], page 34, hereof.

Software means the EVL QVD Utils, as a software for data conversion of QVD file format to CSV format or other data formats and vice versa from those formats to QVD format provided by the Licensor, as it may be later updated, upgraded, amended or otherwise modified and/or renamed or rebranded, including all complementary or derivative software works or programs provided by the Licensor under these Terms, whereas the Software shall be also interpreted as an executable object code of the Software including all its updates, patches, modifications, enhancements, designs, concepts, documentation or other materials that constitute the Software.

EVL QVD Utils Edition
has the meaning stated in Section [License], page 30, of these Terms.

Fee means a remuneration for granting of the EVL QVD Utils License by the Licensor to the Licensee in the amount stated in the Price list or in the written agreement concluded by the Parties.

Installation
has the meaning stated in Section [Installations], page 30, of these Terms.

Intellectual Property Rights
means current and future worldwide rights under copyright laws, trade secret laws and laws to know-how, trademark laws, patent laws and other similar rights.

License means right to use the Software under the terms and conditions set in these Terms, as well as under other applicable documents (if concluded).

Licensee means a person or entity using the Software under these Terms, i.e. unless other specific license terms or agreement states otherwise.

Licensor	means EVL Tool, s.r.o. or other entity to which EVL Tool, s.r.o. may transfer or license all licensable rights to the Software.
Parties	means jointly the Licensor and the Licensee.
Price List	means the document published on the Licensor's website `https://www.evltool.com/` that states the remuneration to be paid for the License and usage of the Software.
Support	has the meaning stated in Section [Support], page 32, of these Terms.
Terms	means these *Software License, Support and Services*, as they may be later amended.
Third-Party Software	means software or other code distributed subject to licenses from third-party suppliers.

Use of software

General License

Subject to the terms and conditions set in these Terms the Licensor grants to the Licensee a limited, revocable, non-exclusive, non-transferable, non-sublicensable license to download, install and use the Software with restrictions as set further below in these Terms.

Installations

Installation means the production installation of the Software (the 'Installation'). Licensee is not limited by the number of testing, development or other types of installations assuming at least one is in production. For running on virtual machines or containers production the Installations means running in parallel as a part of different business process (in production). Each Installation of the Software shall be interpreted as a separate License, except to the Installation on multiple nodes for the same data processing which is considered as one Installation.

EVL QVD Utils Trial License

The Software may be used by the Licensee and/or his/its Affiliates during a **6 weeks trial period** starting on the date of first download of the Software. After expiration of the trial period the Software may be used only under the EVL QVD Utils License as defined in Section [License], page 30.

EVL QVD Utils License

For the Software usage the Licensor shall grant the License to the Software to the Licensee for the Fee (the 'EVL QVD Utils License'). The EVL QVD Utils License shall be granted as one out of three editions:

- Standard
- Premium
- Enterprise

The editions differ by level of the support as defined in Chapter [Technical Support and Services], page 31, and by the number of Installations.

Standard and Premium grants Licensee License to the 1 Installation, while Enterprise Edition limits the number of Installations in the Annex of this Terms.

The EVL QVD Utils License is granted for a period of one year or 3 months depending on the option chosen during the Software purchase. Should not any Party notify the other Party in writing or by email30 days before the end of the period that the notifying Party is not interested in prolongation, the EVL QVD Utils License shall automatically prolong for another period.

Restricted Use

The Software and all associated Intellectual Property Rights remain the property of the Licensor. Unless specifically approved otherwise by the Licensor in writing the Licensee cannot

a. copy, modify, adapt, translate or otherwise create any derivative works of the Software,
b. reverse engineer, decompile or otherwise attempt to discover the source code of the Software or encourage or support anyone else to do the same,
c. sell, rent, lease, assign, sublicense or otherwise transfer any rights to the Software to third parties,
d. distribute, market or otherwise commercialize the Software,
e. remove any copyright, trademark, proprietary notice or label placed on the Software or attached to it or
f. use or introduce any device, software or routine which interferes or attempts to interfere with the operation of the Software.

In addition to the above, the Licensee acknowledges that the CCTZ software may have been used while developing the Software and stipulates to fully and unconditionally respect also the applicable license terms related to the CCTZ software available at `https://github.com/google/cctz/blob/master/LICENSE.txt`.

Registered Use

Each Licensee may be required to register itself as a user or customer before or after downloading the Software. The Software may be installed and used only on a computer controlled by the individual and/or entity specified in the User/Customer Registration Form or other relevant form filled as a prerequisite for the download of the Software or any time thereafter, as the case may be.

Right of Audit

The Licensor may ask the Licensee to provide information in written form about the number of the installations and Licenses being used and other relevant information regarding the usage of the Software. This request may happen no often than once per year unless evidence of material difference between number of Installations and Licenses exists. The Licensee shall provide the information within 7 days of the delivery of the request of the Licensor to the Licensee.

Unlawful Use

In case of unlawful use of the Software the Licensor shall be entitled to claim against the breaching Licensee all damages and in case of unlawful use of the Software for commercial or other profit-oriented purposes contrary to this Terms moreover a contractual penalty in the amount of EUR 10,000.

Technical Support and Services

New Versions Notification

The Licensee is entitled to be notified of new EVL QVD Utils versions and updates and their relevant terms. Notifications will be sent by e-mail to the respective e-mail address(es) specified in the User/Customer Registration Form or otherwise communicated to the Licensor as e-mail address(es) designated for this purpose.

E-mail Technical Support

The Licensor provides technical support of the Software using dedicated email: `support@evltool.com`.

Incident Management System

The EVL QVD Utils Enterprise Edition License includes right of the Licensee to be provided with the incident management system access.

Service Level Details

Working hours and response time measurement

The Support shall be provided during regular working hours – i.e. during business days in the Czech Republic between 9:00 and 17:00 Central European Time. The response time is defined as a period between the time when the Licensee notified the Licensor about the incident and the moment when the Licensor replied to that notification with confirmation of the incident severity and the next steps proposal. Response time is measured towards the business hours as defined in this article.

Incident classification and response time

Incidents shall be classified into 3 severities:

a. severity 1 – production installation of EVL QVD Utils does not run and no workaround is available;
b. severity 2 – EVL QVD Utils functionality differs to documentation, but EVL QVD Utils runs at least with workaround; and
c. severity 3 – minor difference of functionality comparing to documentation which does not prevent EVL QVD Utils usage.

Response time differs according to the incident severity and shall be as follows:

- severity 1 – response time 4 business hours
- severity 2 – response time 12 business hours
- severity 3 – response time 24 business hours

Licensor Support Contacts

Any operational or functionality problem shall be notified to the Licensor at the below contact information or at other contact information, as it may be later changed by the Licensor:

a. +420 602 643 752; and
b. `support@evltool.com`.

Technical support limitation

The Support under this Agreement is limited exclusively to the EVL QVD Utils. Incidents related to other parts of the Licensee's IT environment, e.g. operation system configuration, connectivity setup etc. are excluded.

EVL QVD Utils Editions and Service Level

Differences in the service level for EVL QVD Utils Editions defines next table:

Updates and Support Limitation

Updates

Newer versions and updates of the Software shall be considered as part of the Software and may be used by the Licensee under the same terms and conditions as set in these Terms with respect to the Software unless the Licensor resolves later otherwise.

Support Limitation

The Licensor shall not be bound to provide any support or any kind of further assistance with respect to the Software beyond as specified in Section [Technical Support and Services], page 31.

License Fees

EVL QVD Utils Trial License

EVL QVD Utils Trial License is provided free of charge.

EVL QVD Utils License

The EVL QVD Utils License shall be granted and provided for the Fee. The Licensee shall pay the Fee to the Licensor based on Licensor's invoice within 15 days as of the first day of the respective one-year period for which the EVL QVD Utils Enterprise Edition License is granted. Each invoice will be surcharged by the applicable value added tax in compliance with the relevant legislation. Should the License be terminated before the expiration of the period that the License was agreed for, the Fee will not be refunded.

Limited liability of the licensor

Disclaimer

Unless stipulated otherwise in a separate written agreement the Licensor does not guarantee functionality and/or operability of the Software and shall not be held liable for any failure, inoperability or other deficiencies of the Software and/or any damages caused by the Software or in association with it.

Warranty Disclaimer

The Software is provided on an '*as is*' or '*as available*' basis without any representations, warranties, covenants or conditions of any kind. The Licensor and it suppliers do not warrant that the Software will be free from all bugs, errors or omissions. The Licensor and its suppliers disclaim any and all other warranties and representations with respect to the Software, including any and all (i) warranties of quality and functionality, (ii) warranties of design and (iii) warranties of suitability for any purpose.

Licensee's representation

Licensee represents, warrants and covenants that: (a) all of its employees and consultants will abide by the terms of this Terms; (b) it will comply with all applicable laws, regulations, rules, orders and other requirements, now or hereafter in effect, of any applicable governmental authority, in its performance of this Terms. Notwithstanding any terms to the contrary in these Terms, Licensee will remain responsible for acts or omissions of all employees or consultants of Licensee to the same extent as if such acts or omissions were undertaken by the Licensee.

Designed Purpose

The Software has been developed for general use in variety of information and data management applications. The Licensor does not guarantee suitability of the Software for any specific purpose.

Liability Cap

Unless stipulated otherwise in a separate commercial agreement, the Licensor shall not be held liable for any damages caused by the Software or in relation to it or by the Licensor under these Terms unless the harm is caused to the natural rights of an individual or is caused intentionally by the Licensor or due to gross negligence of the Licensor.

Indemnification

Licensee will indemnify, defend and hold EVL Tool Indemnitees harmless from and against any and all actual or threatened suits, actions, proceedings, claims (groundless or otherwise), damages, payments, deficiencies, fines, judgments, settlements, liabilities, losses, costs and expenses (including, but not limited to, reasonable attorney fees, costs, penalties, interest and disbursements) resulting from any claim (including third-party claims), suit, action, or proceeding against any EVL Tool Indemnitees, whether successful or not, caused by, arising out of, resulting from, attributable to or in any way incidental to:

a. any breach of these Terms (including, but not limited to, any breach of any of Licensee's representations, warranties or covenants);
b. the negligence or wilful misconduct of the Licensee; or
c. the data and information used in connection with or generated by the use of the Software.

Termination

Termination

The Licensor may terminate the License to the Software provided under these Terms with an immediate effect if material breach of these Terms or applicable contractual documents or legal norm occurs or if the Licensee is declared bankrupt or is in liquidation. A default of the Licensee with a payment of the Fee longer than 30 days shall be considered as the material breach. The termination shall be done by a written notice or notice delivered by e-mail specifying the respective termination reason. The termination date shall be the date when the termination notice is delivered to the Licensee.

The Licensor may terminate the Community Edition License without any reason. The termination shall be done by a written notice or notice delivered by e-mail. The termination date shall be the date when the termination notice is delivered to the Licensee.

Effect of Termination

Upon termination of the License, the Licensee shall be bound to immediately stop using the Software, uninstall it from all computers under his/its control and to delete or otherwise destroy all files containing the Software or its part. Section [Unlawful Use], page 31, of these Terms applies accordingly.

Confidentiality and data protection

Confidential Information

All non-public information relating to the Software and/or the Parties shall be considered confidential (the **'Confidential Information'**). Confidential Information relating to the Software shall be considered as Confidential Information of the Licensor.

Confidentiality Obligations

Each Party will protect the other Party's Confidential Information with reasonable care. Each Party may disclose the Confidential Information to third parties only (i) with prior approval of the other Party, (ii) if required by law or (iii) to his/its representatives, employees and/or advisors who need to know it and who have agreed in writing to keep it confidential.

Publicity

Neither Party will issue any press release, public announcement or other statement regarding the existence or content of the License; provided, however, the Licensor may use the Licensee's brand features (primarily name and logo) in its marketing (primarily as a reference) unless the Licensee explicitly informs the Licensor in writing that he/it disagrees with such publicity.

Personal Data

At the moment of conclusion of this Terms between the Parties processing of personal data by the Licensor is commenced. Processing of the personal data is regulated by the principles regarding the protection of personal data published on the Licensor's website `https://www.evltool.com/`.

Miscellaneous

Amendments

The Licensor may amend these Terms or Price List unilaterally. Amended Terms or the Price List shall be published on the website `https://www.evltool.com/` and the Licensee shall be notified about the amendment by email at least one month before the effectiveness of the amendment. In case of Licensee's disagreement with the amendment, the Licensee is entitled to terminate the Terms in writing or by email within one month of the delivery of the notice to the Licensee. The Terms are terminated after one month from the delivery of the notice of termination to the Licensor.

Notices

All notices to the Licensor shall be sent to Licensor's address registered in the Commercial Register or to email `license@evltool.com`.

Assignment

The Licensor is allowed to transfer, assign or license all licensable rights to the Software to any third party without consent or approval of the Licensee. Transfer, assignment or license according to the previous sentence will be notified to Licensee by the same way as modification of these Terms according to Section [Amendments], page 35, hereof. The Licensee may not assign, transfer or sublicense the License or any rights under these Terms to any third party without a prior written or e-mail approval of the Licensor.

Entire Agreement

Unless agreed otherwise between the Parties these Terms shall constitute the entire agreement of the Parties with respect to the subject matter hereof and supersedes all prior agreements with respect to the subject matter hereof.

Severability

Should any provision of these Terms be found void, invalid or unenforceable, such provision shall not affect the other provisions hereof and the Parties shall be bound to approve modification of these Terms so that the original intention is retained.

Survival

Notwithstanding termination of the License or any part of these Terms, any provision of these Terms that by their nature are intended to survive, e.g. contractual penalty and/or damages claims, will survive the termination.

Governing Law and Jurisdiction

These Terms shall be governed by the laws of Czech Republic (excluding conflicts of law principles). Any dispute arising out of these Terms or in relation with it shall be subject to the exclusive jurisdiction of the Courts of the Czech Republic.

Conflict of Terms

If there is a conflict between the documents applicable with respect to the Software and/or its use, the documents will be applied in the following order: any specific agreements (made or confirmed by the Parties in writing or via e-mails), separate commercial or other terms, these Terms.

Effectiveness of Terms

These Terms shall become valid and effective on the day, when the first of the following occurs: (i) the Licensee accepts the Terms via the Internet, (ii) both the Parties sign the Terms, (iii) the agreement, whose part are these Terms, is concluded between the Parties or (iv) the Licensee download the Software.

Variable Index

Data Type Index