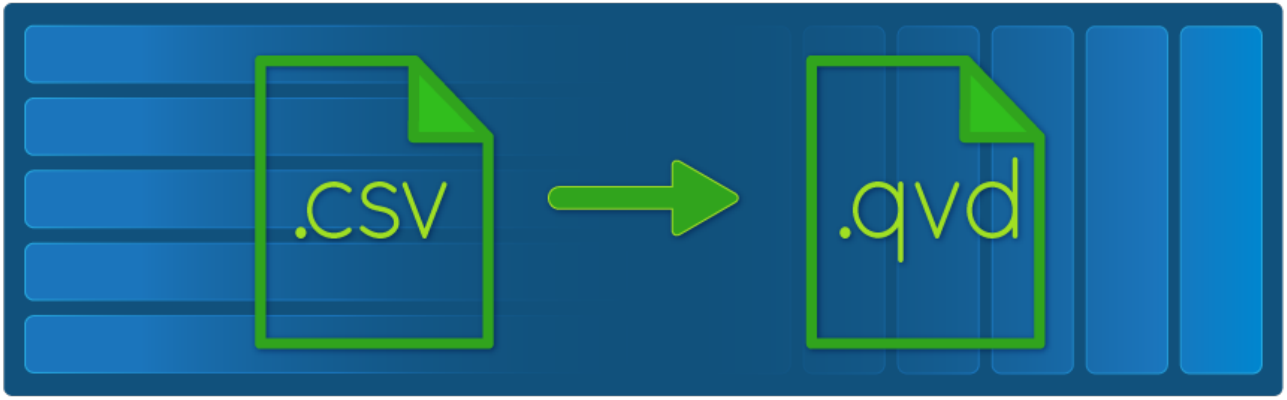
QVD Writer Manual



QVD Writer Version: **0.1**

This document version: 2019-08-12

EVL Tool products

- **EVL** (Extract–Validate–Load) – code based ETL tool (= Extract–Transform–Load);
- **EVL Worfklow** – code based orchestration tool;
- **EVL File Registration** – manipulate metadata about file processing;
- **EVL Manager** – graphical web interface to inspect and manage job and workflow runs, particularly useful for operations.

EVL Microservices

Microservice is a particular EVL functionality together with predefined jobs which generate jobs based only on a configuration CSV file.

- **Anonymization**
- **Data Generation**
- **Validation**
- **Key Generation**
- **Historization**
- **Staging**
- **ASN.1 decoder**
- **QVD Writer**

- **Translation Microservice** – translate file formats or tables

From file/table		To file/table
CSV		CSV
JSON/XML		JSON/XML
XLS/XLSX		XLSX
QVD/QVX (Qlik)		QVD/QVX (Qlik)
Avro/Parquet		Avro/Parquet
Impala/Hive	⇒	Impala/Hive
Oracle		Oracle
Teradata		Teradata
PostgreSQL		PostgreSQL
MariaDB/MySQL		MariaDB/MySQL
mdb (MSSQL)		mdb (MSSQL)
any ODBC		any ODBC

EVL philosophy

- Component oriented with clear focus:
  "**Do One Thing and Do It Well**".
- Do not let simple things to get complex.
- Keep good balance of robustness and functionality.
- Templates and variables oriented – high level of abstraction.

QVD Writer – an EVL Microservice

EVL Microservices are built on top of the core EVL software and retain its flexibility, robustness, high productivity, and ability to read data from various sources; including csv and Excel files, databases – Oracle, Teradata, SQL Server, etc – and Hadoop streaming data like Kafka.

QVD Writer is an EVL Microservice which enables writing into QVD files without using Qlik Sense or QlikView. Very useful when you need to load data from various sources immediately into QVD file and/or you don't want to stuck your Qlik's server by creating many and large QVD files.

History

Version	Date	Brief description
0.1	2019/10	Initial version of QVD Writer.
		New utilities: csv2qvd, csv2evd
		New features: supported Qlik Sense data types: string, date, timestamp, time, numeric, integer
0.2	2020/04	
		Changes:
		New features:
0.3	2020/10	
		Changes:
		New features:
0.4	2021/04	
0.5	2021/10	

Table of Contents

1 Installation

For simple usage, like writing QVD from CSV file, no EVL installation is needed. For more complex ETL processing, like writing into QVD based on JSON for example, EVL would be needed.

1.1 GNU/Linux, macOS

1. For simple usage, like writing QVD only from CSV file, no EVL installation is needed.
2. Login as dedicated technical user.
3. Copy `qvd_writer.tgz` into `$HOME` (or any other suitable target) and unpack it:

```
cd           # or change to any other suitable target
tar xzvf qvd_writer.tgz
```

4. Inside `qvd_writer` folder are then following scripts and binaries:

```
csv2evd
csv2qvd
core/evl_write_qvd
core/csv_to_evd
```

1.2 Windows

1. For Windows, only simple usage, like writing QVD only from CSV file, is available.
2. Login as dedicated technical user.
3. Copy `qvd_writer.zip` into some suitable target and unzip it.
4. Inside `qvd_writer` folder are then following binaries:

```
csv2evd.bat
csv2qvd.bat
core/evl_write_qvd.exe
core/csv_to_evd.exe
```

2 csv2qvd

Available since version 0.1.

Description

Read <file.csv> or STDIN, guess data types or use <file.evd> and write QVD file to <file.qvd> or STDOUT.

Usage

```
csv2qvd
  [<file.evd>|-d <inline_evd>] [-i|--input=<file.csv>] [-o|--output=<file.qvd>]
  [-h|--header=<field_name>,...] [-n|--no-header]
  [-q|--quote=<char>] [-s|--separator=<char>]
  [--date=<date_format>] [--timestamp=<date_format>]
  [-v|--verbose]

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

Options

EVD options:

`--date=<date_format>`
: by default it tries only `%Y-%m-%d`, then `%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 (use -n option then) or when other field names sould be used

`-i, --input=<file.csv>`
: read input <file.csv> instead of STDIN

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

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

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

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

`--timestamp=<date_format>`
by default it tries only `%Y-%m-%d %H:%M:%S`, then `%Y%m%d%H%M%S`

QVD options:

`-d, --data-definition=<inline_evd>`
either this option or the file `<file.evd>` must be presented

`-a, --dos-eol`
suppose DOS end-of-line, i.e. replace CR+LF (`\r\n`) by LF (`\n`) on input

`-b, --mac-eol`
suppose Mac end-of-line, i.e. replace CR (`\r`) by LF (`\n`) on input

`-s, --skip-bom`
skip BOM, i.e. remove 3 Bytes from the beginning of input. These 3 Bytes are usually inserted for UTF-8 files produced on Windows.

`-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

2.1 Examples

1. Following invocation will guess data types, field separator and if strings are quoted or not (all by using csv2evd), and use header line for field names:

```
csv2qvd < table.csv > table.qvd
```

With the `--verbose` option it will write to STDERR the whole EVD file which was used:

```
csv2qvd --verbose < table.csv > table.qvd
```

2. To skip header and use different field names:

```
csv2qvd --header="first_field,other_field,last_one" < table.csv > table.qvd
```

3. 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
```

4. No header in CSV and use generated field names `field_001`, `field_002`, ...:

```
csv2qvd --no-header < table.csv > table.qvd
```

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

```
csv2qvd --date="%j" -s '|' < table.csv > table.qvd
```

6. To use own (specific or already generated) EVD file (i.e. data types definition):

```
csv2qvd table.evd < table.csv > table.qvd
```

3 csv2evd

Available since version 0.1.

Although csv2qvd can call this utility automatically, for production usage it would be better to have data types firmly in our hands. For such purpose it might be good to produce estimated EVD files (EVL data types definition) automatically. And that is the aim of this utility.

Description

Read `<file.csv>`, guess data types, field separator, if strings are quoted, and write EVD to STDOUT or to `<file.evd>`.

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

Separator is trying to be guessed in this order: `,`, `;`, `:`, `\t`, `SPACE`, `|`.

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

Usage

```
csv2evd
  <file.csv> [-o|--output=<file.evd>]
  [--date=<date_format>]
  [-h|--header=<field_name>,...]
  [-n|--no-header]
  [-q|--quote=<char>]
  [--separator=<char>]
  [--timestamp=<date_format>]
  [-v|--verbose]

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

Options

`--date=<date_format>`
: by default it tries only `%Y-%m-%d`, then `%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 or when other field names would be used

`-n, --no-header`
with this option it suppose there is no header. Fields will be named field˙001, field˙002, etc.

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

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

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

`--timestamp=<date_format>`
by default it tries only `%Y-%m-%d %H:%M:%S`, then `%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

3.1 Examples

1. Following invocation 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 file (i.e. EVL data definition file):

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

2. To skip header and use different field names:

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

3. 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
```

4. No header in CSV and use generated field names `field_001`, `field_002`, ...:

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

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

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

4 Examples

4.1 Simple usage 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 `STDERR` used EVL data types description:

```
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"
---------
```

4.2 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"
```