# Installation and users Guide for the reference implementation

## Licence and Credits



# Introduction

The reference implementation serves the following goals:

- Test and prove the concept of the DVPD
- Provide reference about the transformation ruleset for other implementations
- Provide examples for model and mapping variation
- Provide examples / templates for code and documentation generators

It is implemented completly in python and provided under the Apache 2.0 licence:
http://www.apache.org/licenses/LICENSE-2.0

# Architecture and content of the reference implementation

The reference implementation is part of the DVPD git repositiory. It consists of the following assets:

- DVPD Compiler (DVPDC) (processes/dvpdc/*_main_*.py)
- Some assisting libraries (lib/*.py)
- Set of dvpd testcases and example dvpd (testcases_and_examples/dvpd/*.dvpd, testcases_and_examples/model_profiles/*)
- Automated test of the DVPDC (processes/test_dvpdc/*_main_*.py)
- Reference results for the automated testing ((testcases_and_examples/reference/*.dvpd))
- Example of a ddl Render script (processes/render_ddl/*_main_*.py)
- Example of a documentation render script (processes/render_documentation /*_main_*.py)

The scripts are using a central configuration file (dvpdc.ini file), to declare further directory structures (See "Decistions about file locations" below)

- template for the configuration file (config_template/dvpdc.ini)

# DVPD usage Worflow

The general workflow, would be as follows:

1. Generate and edit the dvpd document of the desired pipeline

2. store the dvpd in the directory for dvpd's in your project
3. Compile the dvpd. This will result in
    - a log output with compiler messages on console and the reporting directory
    - a dvpi file in the desigated dvpi directory
    - a dvpi summary report in the reporting directory
4. review the compiler messages and the dvpi summary. If you need to correct mistakes, loop back to step 3
5. run the ddl render script for the new generated dvpi to generate the ddl files in the repository for ddl files
6. deploy the data model to the data base (using the ddl files and the deployment procedure of your choice). This will test the db compatibility of your ddl files.
7. probably commit dvpd, dvpi and the ddl files in a git repository of the project
8. generate other assets from dvpd and dvpi (e.g. documentation, loading code, etc.)
9. finalize the implementation of the pipeline

Feel free to integrate this into a CI/CD workflow of your choice.

# Installation Guide

## Requirements

- python 3.10 or higher must be available
- please install missing python packages on demand (via python pip or equivalent package manager)
- A text editor (hopefully capable of JSON syntax highlitging and hierarchie folding)

If you want to modify, debug or extend the dvpd toolset or documentation

- An text editor soupporting markdown documents
- "Draw.io" for opitmal view of diagrams
- A python ide

## Decisions about file locations

You need to decide, where to place the following artifacts

- the DVDP project, that includes the compiler
- configuration file(=ini file) for the compiler
- your DVPD files (probably inside of a git repository of your project)
- your "model profile" configuration files (probably inside of a git repository of your project)
- compiler results files (Reports, dvpi files)
- results from generators (e.g. the generated DDL files should also go into the git reposotory of your project)

Example structure

```
\dvpd_compiler            <- the compiler project
\dwh_resources            <- the git repository of your dwh project
     \dvdc_config         <- dvpdc ini file
```

```
      \dvpd_model_profiles  <- dvpd model profiles
      \dvpd                 <- dvpd files
      \dvpi                 <- dvpi files
      \model_ddl            <- genrated DDL files
\var\dvpdc_report           <- log output of dvpdc
```

## Download and setup the compiler

- Download or clone the DVPD repository in the directory for the DVPD project
- copy the file "dvpdc.ini" from the "config_template" directory of the project to your desired location for configuration files
- adapt all properties of the [dvpdc] section in the "dvpdc.ini" file entries to point to the desired directories
- adapt all properties "ddl_root_directory" and "dvpi_default_directory" of the [rendering] section in the "dvpdc.ini" file entries to point to the desired directories
- copy the file "default.model_profile.json" from "testset_and_examples\model_profiles" to your desired location for model profiles
- adapt the model profile to your project needs (see Reference_of_model_profile_syntax.md)
- add the directory "/commands" from the compiler project to your path environment variable
- open a command line and run "dvpdc -h" in any directory. This should show the help text of the compiler

### Test the compiler

- place a dvpd file in the directory for the dvpd files
- open a command line
- run "dvdpc --ini_file="<path to the ini file>"
- This should write out its messages and success state to the console and a log file in the log output directory
- if the compile is successfull you should have a dvpi file in the directory for dvpi

### Test the ddl generator

- run "dvdp_ddl_render <name of the dvpi file> --ini_file="<path to the ini file>"
- all ddl scripts for the pipeline in the dvpi file should be written to subdirectoreis of th configured "ddl_root_directory"

### Test the documentation generator

- run "dvdp_doc_render <name of the dvpd file> --ini_file="<path to the ini file>"
- a html file, containing a formatted the mapping tabl should be written to subdirectoreis of the configured "documentation_directory"

## Create some shortcuts or convinience wrapper

Since declaration of the ini files will mostly be the same in every call, you might want to create some wrapper script, that inserts this

This wrapper script might also be helpfull to concatinate some steps of your workflow into one call.

## Adapt the ddl generator

The ddl generator is only an example/template and must be adapted to your specific need, regarding SQL dialect or naming of files and directories.

Please be aware, that many design descisions about your platform can be adjusted in the model profile (e.g. names and types of metadata and hash columns) and will already be applied to the compilers result in the dvpi. Only the final formatting, sorting and SQL Syntax of the ddl files lies in the responsibility of the ddl generator.

To adapt the ddl generator it is recommended to copy the template and make it your own code.

# Usage Guide

## dvpd compiler (dvpdc)

The dvpdc compiler is started on the command line with

```
dvdpc <name of the dvpd file> options
```

When using the file name "@youngest", the compiler uses the youngest file in the dvpd default directory

Options:

- --ini_file=<path of ini file>:Defines the ini file to use (default is dvpdc.ini in the local directory)
- --model_profile_directory=<directory>: Sets the location of the model profile directory (instead of the location configured by the ini file)
- --dvpi_directory=<directory>: Sets the location for the output of the dvpi file (instead of the location configured by the ini file)
- --report_directory=<directory>: Sets the location for the log and report file (instead of the location configured by the ini file)
- --verbose: prints some internal progress messages to the console
- --print_brain: prints the internal "memory" of the compiler

### result

The compiler creates a log file and, when compilation is successfull 2 result files

- report directory
  - log file: Contains the same messages, that have been written to the console
  - dvpisum.txt: Contains a summarized version of the dvpi data for fast overview about all essentials, created by the compiler
- dvpi directory:
  - dvpi file: Contains the dvpi json file, that has been generated from the dvpd. This can be used as a base for all further generators (see Reference_and_usage_of_dvpi_syntax.md)

## ddl generator (dvpd_ddl_render)

The ddl generator renders all create statements for a given dvpi file. Since this is an example and not core part of the dvpd concept, syntax and structure are also only example (mainly taken from a current project). To

adapt this to your needs, you should copy and adapt the script.

The ddl generator is started on the command line with

```
dvdp_ddl_render <name of the dvpi file> options
```

When using the file name "@youngest", the script uses the youngest file in the dvpi default directory

Options:

- --ini_file=<path of ini file>:Defines the ini file to use (default is dvpdc.ini in the local directory)
- --print: prints the full ddl set to the console
- --add_ghost_records: adds ghost record inserts to the script
- --no_primary_keys: omit rendering of primary key constraints
- --stage_column_naming_rule={stage|combined} : stage = pure genrated stage column names are used in the stage combined= combination of target column names and stage column names

Settings read from .ini file:

- dvpi_default_directory
- ddl_root_directory
- stage_column_naming_rule

### Purpose of the "combined" stage table generation rule

Some data vault loading frameworks (e.g. cimt talend framework) use similariy of column names between stage table and target table for automatic mapping. In that case it is more convenient to generate a stage table with the target column names. This will work well, until the same column name for different content is used in different tables of the pipeline or different source fields are mapped to the same target.

The "combined" stage column naming rule resolves this as follows:

- one or more target columns with same name and same single source field: stage column name = Target name
- one target column with a unique name and multiple source fields: stage column name = Target name + field name
- multiple target columns with different names have the same single source field: stage column name = all target field names concatinated
- multiple target columns sharing the same name using different source fields: stage column name = field name

## Documentation generator (dvdp_doc_render)

The documentation generator creates a simple html table of the field mapping, ready to be copied into a documentain tool of your choice (confluence, one Note / Word, ...)

The documentation generator is started on the command line with:

```
dvpd_doc_render <name of the dvpd file> options
```

When using the file name "@youngest", the script uses the youngest file in the dvpd default directory

Options:

- --ini_file=<path of ini file>:Defines the ini file to use (default is dvpdc.ini in the local directory)
- --print: prints the html text to the console

## Developer sheet generator (dvpd_devsheet_render)

The developert sheet is a human readable text file, providing the essential key information needed to implement the loading process.

- pipeline name
- record source
- source field structure
- formatted list of tables involved (targets + stage)
- stage table structure and mapping of fields to stage table
- hash columns and how to assemble it
- load operations for every table and the stage column to target column mapping

The developer sheet generator is started on the command line with:

```
dvpd_devsheet_render <name of the dvpi file> options
```

When using the file name "@youngest", the script uses the youngest file in the dvpi default directory

Options:

- --ini_file=<path of ini file>:Defines the ini file to use (default is dvpdc.ini in the local directory)
- --stage_column_naming_rule={stage|combined} : stage = pure generated stage column names are used in the stage combined= combination of target column names and stage column names

see "stage_column_naming_rule" in the ddl generator for explanation of the naming rules.

Settings read from .ini file:

- dvpi_default_directory
- stage_column_naming_rule

## Full compile and render for youngest dvpd (dvpd_all_youngest)

This command call all steps with the "@youngest" directive a file name parameter. Therefore it compiles the youngest dvdp and generates ddls and documentation, when compilation was successfull.

It a first rough example, how to create a ci pipeline.