DVPI Syntax Reference and usage guide

DVPI is the resultset created from the compiler by transforming the a dvpd. It is mainly designed to be read by generators for creating loading code and DDL.

In this document you find a brief guideline on how to use DVPI content in your loading processes and a full syntax reference based on the core DVPD syntax.

Usage guideline

DVPI is designed to support the steps of loading a source object into a data vault model. The following steps are normally needed for the loading:

- deploy the target database tables (might be done by the first run of the job or in sync with the deployment of the job artifact)
- contact the source system and determine the increment
- fetch the data increment from the source system
- parse the fetched data into rows and fields (This might involve identification and translation of deletion events)
- calculate the needed hash values from the fetched data (might be part of every load operation or done in advance by creating a stage table)
- load the parsed data and hashes into the data vault tables in all necessary combinations (this might involve detection of missing source data and its translation in deletion flagging)

The listed steps can be implemented in different approaches as there are

- stage + target : Incoming data and calculated hashes are stored in a stage table first, then loaded to the target tables
- OSD + target: Incoming data is parsed into an "operational data store" (might be in memory only), hashes are calculated on the fly during the load into a target table
- target only: Incoming data is parsed and hashed for every target table on the fly

This guideline describes, what should be used from the DVPI and how to get it. It is by no means a recipe for the finally executed loading code. At least before execution a code generator should create the case and platfom specific instructions (e.g. SQL Statements) that can be executed efficiently.

deploy target database tables

To deploy the target database tables, a ddl sql with the create statement must be generated somehow. The necessary information to generate the ddl can be found in the DVPI as follows:

- iterate over the tables[] list
- inside a table list entry you find

    - schema and table name
    - table stereotype (hub,lnk,sat,ref)
    - iterate over the columns[] list containing
        - column name, type
        - indication of not null contraints
        - column class, can be used to identify specific columns (e.g. Business keys) that are interesting for indexes or primary key constraints depending on the table stereotype

Should your loading pattern involve **stage tables** you also need to retrieve the stage table structures as follows

- iterate over the "parse_sets[]" list
- inside a parse set entry you find
    - stage_properties [] list, containing stage table schema and name for every storage component involved
    - stage_columns [] list with
        - stage column name and type
        - flag to determine if the column is nullable

contact the source and determine the increment

This should be clarified by all properies in the **"data_extraction"** section. It depends heavily on your implementation architecture and module flexibility, how far the behavior of the fetching process can be configured. Therefore this guide can only forward you to the documentation of your modules.

fetch the data increment from the source

This also is mostly defined by the properties in the "data_extraction" section.

iterate over parse sets

Depending on the data source, there will be one or more parse sets for the fetched data. Tabularized (DB, CSV) sources have only one set, whereas hierarchical data (JSON, XML) structures with multiple loops will have more. (upcoming feature in DVPD 0.7.0). Every parse set needs have its own **definitions for parsing, hashing/staging and loading**. This is why all declarations for this and the following steps **are subelements in the list of parse_sets**.

parse the fetched data into rows and fields

To parse the data for a single parse set, iterate through the **fields[]** list.

At least the following properties will be available - field name and type - field position The field name is essential, since it is used as the identifier in the mappings.

More properties depend on the needs for the parsed format and the capabilities of the parsing module.

The parsed data row might be directly stored in the ODS with the field_name as column name or kept in memory for staging. (It is not recommended to use the stage column name in the ods. This prevents confusion about the used pattern)

load data into stage table

This is only needed, when using the stage+target approach.

- interate over hashes[] list
    - if there is a hash with "multi_row_content" = true execute necessary steps to provide multi row hashes (diff hashes of multi active satellites)

To stage a single row

- iterate over **stage_columns[]** list. Depending on the stage_column_class...
    - **"meta..."** put in the appropriate value (e.g. record_source_name_expression, deletion flag)
    - **hash** calculate the hash value according to the recipe provided by the "hashes[]" list entry with the same "hash_name" and its "hash_fields[] list
    - **data** put in the value of the field, probably with transformation of the data type

load data from stage to the target tables in all possible combinations

This only applies, when using the stage+target approach.

- in the current parse set, iterate over load_operations[] list
    - retrieve table stereotype and properties by looking up the table name in the tables[] list.
    - execute the loading steps for the table stereotype
        - iterate over all meta fields from the tables columns[] list and provide necessary values accordingly
        - the data mapping must be read from the "hash_mappings[]" and "data_mappings[]" lists
            - copy data from the stage column "stage_column_name" to the target column "column_name"
            - use the column_class to identify columns, that have special meanings in the loading (business keys, diff hash, etc)
        - for satellites
            - follow driving key directive
                - follow deletion detection rule

load data from Ods to the target tables in all possible combinations

This only applies, when using the ods+target approach.

- in the current parse set, iterate over load_operations[] list
    - retrieve table stereotype and properties by looking up the table name in the tables[] list.
    - execute the loading steps for the table stereotype
    - the mapping of data columns must be read from "data_mappings[]" lists
        - copy data from the ods column "field_name" to the target column "column_name"
    - the mapping of hash columns must be read from the "hash_mappings[]" lists
        - calculate the hash value according the the recipe provided by the "hashes[]" list entry with the same "hash_name" and its "hash_fields[] list

    - store the value in the column "column_name"
  - use the column_class to identify columns, that have special meanings in the loading (business keys, untracked)
  - iterate over all meta fields from the table columns[] list and provide necessary data accordingly
  - for satellites
    - follow driving key directive
    - follow deletion detection rule

load data directly from source to the target table

- in the current parse set, iterate over load_operations[] list
  - retrieve table stereotype and properties by looking up the table name in the tables[] list.
  - execute the loading steps for the table stereotype and the declared deletion detection procedure
  - the mapping of data columns must be read from "data_mappings[]" lists
    - lookup the fields[] entry for the "field_name"
    - parse the field content from the source, by following the declaration of parsing properties
    - store the value in the column "column_name"
  - the mapping of hash columns must be read from the "hash_mappings[]" lists
    - calculate the hash value according the the recipe provided by the "hashes[]" list entry with the same "hash_name" and its "hash_fields[] list
    - store the value in the column "column_name"
  - use the column_class to identify columns, that have special meanings in the loading (business keys, untracked)
  - iterate over all meta fields from the tables columns[] list and provide necessary data accordingly
  - for satellites
    - follow driving key directive
    - follow deletion detection rule

Syntax Reference

A DVPI is expressed with JSON syntax and contains the following attributes (Keys):

Root

dvdp_compiler
Textinformation about the compiler and its version

dvpi_version
Version of the dvpi format. In general it is in sync with dvpd version .

compile_timestamp
Compile time. Just for auditibilty

dvpd_version
Version of the used dvpd syntax

pipeline_name
Name of the pipeline, as declared in the dvpd. Could/should be used to identify the loading process artifact(s)

pipeline_revision_tag
String to identify the revision of the pipeline description. Content depends on process and toolset for development and deployment. Might be a version number or a revision / build tag.
Set to '--none--' if not given
Examples: *"1.1"* | *"x129sa8"*

dvpd_filemame
Name of the compiled DVPD file. Just for auditability

tables[]
List of all data vault tables, loaded by this pipeline.
→ see "tables[]"

data_extraction
→ see "data_extraction"

parse_sets[]
List of parsing sets. Currently there will be only one entry. This structure element is already in the DVPI syntax to allow multiple parse sets, when supported by DVPD.
→ see "parse_sets[]"

tables[]

Json Path: $

The main purpose of the tables section, is to provide all structural information, needed to create the model tables, determine the loading procedure and document the relational structure

table_name
Identification of the table in the DVPD Data Model. It is currently also used as name of the database table. This will be separated in later releases.

table_stereotype
Data Vault table stereotype "hub","lnk","sat","ref". Should be used to document the data vault model structure and determine the loading procedure.

schema_name
Database schema of the table. (or database name, when the DB Engine does not support schemas, but uses "Databases" as structuring element)

storage_component
Identification of the storage component. Valid values depend on the system architecture and may control

retrieval of connection parameters and use of platform technology specific SQL Dialect, and loading procedures.

driving_keys[] (optional, and only set when needed on satellites of links)
will be implemented in 0.6.1
List of the hub keys in the parent link of the satellite, that identify the driving objects = Objects, where the complete relation data, expressed by the parent link, is in the currently staged dataset
Without this declaration, no driving key logic should be applied.

has_deletion_flag
Triggers a loading procedure to manage a deletion flag, when processing deletion data

is_effectivity_sat
Indicates to the loading procedure, that there are no data columns to be compared

is_enddated
Indicated to the loading procedure, that an enddate has to be managed

is_multiactive
Indicates to the satellite loading procedure to follow the multiactive satellite loading pattern

compare_criteria
Defines the elements that have to be compared, when loading a satellite. Rows will be loaded when:

- key = the key (hub key, link key) is not already in the satellite
- data = the value combination of the relevant compare columns or the diff hash are not already in the satellite
- current = the value combination of the relevant compare columns or the diff hash are not equal to a current row in the satellite
- key+data = comparison of values is reduced to all rows which share the same key
- key+current = comparison of current values is reduced to the key (this is the main mode of data vault satellites)
- none = data will always be inserted (prevention of duplication from repeated loads must be solved by load orchestration)

uses_diff_hash
Indicates the existence and usage of a diff hash, for comparison of satellite data during the loading

table_comment
Contains the table comment, that should be added to the table in the database

columns[]
→ see "columns[]"

columns[]

Json Path: $.tables[]

column_name
Name of the column in the table. Should be used in DDL generation.

column_type
Datatype of the column in the table. Should be used in DDL generation.

column_comment
Comment provided for the column. Should be used in DDL generation.

is_nullable
Declares if the column should be nullable. Is set to true for all hashes and metadata, since these should never be null.

prio_for_column_position
Criteria to arrange columns in a defined order in the DDL statement.

column_class
Information about the kind of data from perspective of the data vault method. It should be used during DDL generation, when indexes or primary key contraints are generated, to identify the columns of interest.

Possible values:

- **key** - the column is the hub key or link key of a hub or link
- **parent_key**- the column is the key of the parent table of a satellite or link
- **business_key** - the column is part of the business key of a hub
- **content** - the column contains data of a satellite
- **content_untracked** - the column contains data, that does not participate in keys or comparison
- **dependent_child_key** - the column is a dependent child key in a link
- **diff_hash** - the column contains a hash over all relevant columns/rows for comparison during the loading of satellites or reference tables
- **meta_**...
    - meta_load_date
    - meta_load_process_id
    - meta_record_source
    - meta_deletion_flag
    - meta_load_enddate

parent_key_column_name
Name of the key column in the parent table, this column can be joined with. Provided to document the model structure or generate join SQL snippets.

parent_table_name
table_name of the parent table, this column is used to join with. Provided to document the model structure or generate join SQL snippets.

exclude_from_change_detection
Defines, if the column should be used in the change detection for loading satellites or reference tables. (#the meaning needs more clarification, since column_class and uses_diff_hash also control the elements involved#)

data_extraction

Json path: $

Contains all declarations needed to define the methods, how to retrieve the data. This might be just a module name, since it depends highly on the source technologiy and format and the flexibilty of the fetch and parse module.

fetch_module_name
The name of the module to be used for fetching

parse_module_name
(future version)
The name of the module to be used for parsing

...

parse_sets[]

Json path: $

Contains a list of parse sets. Until DVPD version 0.7.0 there will be only one element.

stage_properties
List of stage properties, for different storage comnponents.
→ see "stage_properties[]"

record_source_name_expression
The record source name expression for the current parse set. Might contain placeholders, when the processing module is capable of adding runtime specific information.

fields[]
List of fields and their parsing properties.
→ see "fields[]"

hashes[]
List of hash "recipes" needed.
→ see "hashes[]"

load_operations[]
List of load operations needed.
→ see "load_operations[]"

stage_columns[]
List of stage columns and their field / hash mapping.
→ see "stage_columns[]"

stage_properties[]

Json Path: $.parse_sets[]

List of stage properties, for different storage comnponents. As long as there is only one database system in a data warehouse platform, this will be only one entry. Nevertheless, in case the data model is distributed over multiple storage components and technologies, this provides the opportunity to declare different names for every storage component.

stage_schema
Name of the schema, the stage table will be placed in.

stage_table_name
Name of the stage table.

fields[]

Json Path: $.parse_sets[]

Provides all necessary declarations how to parse every field from the source data.

field_name
Name of the field. This will be used to identify the field in the mappings.

field_type
Datatype of the field. Depending on the parsing module, this might be the type of the incoming data. A type deviation to the target column will result in a type conversion.

field_comment
Comment about the field for pure documentation.

field_position
This is the position of the field in the DVDP field list. It might be relevant for the parsing (e.g. when field order in DVPD represents the order of CSV columns).

field_value
Will be added in release 0.6.2
Allows the declaration of a constant value or a placeholder, that inject data from, that is not directly in the dataset. Valid settings depend on the generator/execution module. General syntax for data placeholder is "${<name of placeholder}". The following placeholders are expected to be available:

- <value> - this value will be taken as field value
- ${CURRENT_TIMESTAMP} - Timestamp of the current load process
- ${ROW_NUMBER_OVER_BUSINESSKEY(list of business keys)} - The numerical position of the row in the defined business key
- ${ROW_NUMBER_OVER_KEY(data vaut key)} - The numerical position of the row in the declared data vault key (e.g. the hub key)
- ${ROW_NUMBER} - The numerical position of the row in source data delivery

Examples for custom placeholders:

- ${DATE_FROM_FILENAME} - parse the name of the processed filename
- ${COMPANY_NAME_FROM_CALL_PARAMETER} - Get the company code from the call parameter

Be aware: This property can be missused to provide cleansing and transformation functions. It is recommended not to do so. It's purpose should be restricted to insert data from the execution environment into the dataset. Every kind of cleansing and transformation of the source data, should be done before the staging.

<more properties to come>

hashes[]

Json Path: $.parse_sets[]

Provides all necessary declarations, how to assemble and calculate the hash values.

hash_name
Identfication of the hash. This is used to identify the hash in the mappings.

stage_column_name
Name of the stage column, containing the hash value.

hash_origin_table
Name of the data vault table, that is the "owner" of the hash (hash is the key of the table or a diff hash in the table)

column_class
Column class of the hash: key , diff_hash

multi_row_content(diff hash only)
Boolean, declaring if the hash has to be assembled from multiple rows. (e.g. a diff hash for multi active satellites)

related_key_hash(diff hash only)
Name of the hash, that defines the the key, the diff hash is referring to. This can be used to identify all rows for the same multi row diff hash. This property is not available, when the key hash of the table is delivered by a source field.

hash_fields[]
List of the fields, that need to be concatenated for the hash.
→ see "hash_fields[]"

column_type
Column type of the hash value in the target table. (This is copied from the model profile, that has to be used for the hash containing table)

hash_encoding
Type of enconding of the hash value. (Binary/BASE64/HEX) (This is copied from the model profile, that has to be used for the hash containing table)

hash_function
Hash function, that has to be applied. (This is copied from the model profile, that has to be used for the hash containing table)

hash_concatenation_seperator
Single character or string, that must be used as seperator between the concatinated fields (This is copied from the model profile, that has to be used for the hash containing table)

hash_timestamp_format_sqlstyle
Format string (using SQL syntax) to convert timestamps into a string representation for concatentation. (This is copied from the model profile, that has to be used for the hash containing table)

hash_null_value_string
String to be used for the data value NULL (Unknown) when concatenating the field data. (This is copied from the model profile, that has to be used for the hash containing table)

row_order_direction
order direction to be used, when assembling data into a group hash. Will be set for multi active satellites.

model_profile_name
This is just for documentation and tracing of the DVPD result and should not be used in any kind of processing of the DVPI.

hash_fields[]

Json Path: $.parse_sets[].hashes[]

The list of fields, to be concatenated for a hash. Be aware of the different properties to tweak the order of the fields for the concatination. The final ordering rule lies in the executing module and must be well documented by it.

field_name
Name of the field, containing the data.

prio_in_key_hash(key hash only)
Priority to be used, for ordering the fields in the concatination for a key hash. This can declared in the DVPD and should have more impact then other field attributes like name or field position.

prio_in_diff_hash(diff hash only)
Priority to be used, for ordering the fields in the concatination for a diff hash. This can declared in the DVPD and should have more impact then other field attributes like name or field position.

prio_for_row_order(diff hash only)
Priority to be used, to define the order the rows, when assembling multi_row_content into a single hash (diff hash for multiactive satellites).

field_target_table
Name of the table, this field is mapped to and causes the participation of the field in the hash. (e.g. name of the hub table when participating as business key of the hub). This should be used to arrange a table name specific order when assembling link keys.

field_target_column
Name of the column, this field the field is mapped to in the target table. This should be used to organize a

column name specific order when assembling link keys. It is highly recommended not to use the field name to create orders, since field names of different mappings might create different orders, where the target column names stay constant.

parent_declaration_position (on link key hashes only)
Only for link hash keys, this is the position of the parent table in the link_parent_tables[] list in the DVPD. This might be used to organize a link parent declaration specific order, when assembling link keys.

load_operations[]

Json Path: $.parse_sets[]

This is a list of all necessary load operations. It contains at least one operation for every target table. Multiple entries for the same target differ in the mapping of fields and hashes and probably deletion detection rules.

table_name
Name of the table to load. Needed to look up all details about the table in the tables[] list.

relation_name
Only for documentation and tracing purpose. This is the relation, the mapping is created for.

operation_origin
Only for documentation and tracing purpose. This describes the rule, that identifed this relation to be loaded for this table.

hash_mappings[]
List of the mappings of all hashes to the table columns.
→ see "hash_mappings[]"

data_mapping
(will be renamed to "field_mappings" in 0.6.3)
List of the mappings of all fields to the table columns.
→ see "data_mappings[]"

deletion_detection_rules[] (optional)
→ deletion_detection_rules[]

hash_mappings[]

Json Path: $.parse_sets[].load_operations[]

column_name
Name of the target column in the table

hash_name
Identification of the hash in the hashes[] list. In case of a direct field mapping, the hash_name will not be set.

hash_class
Class of the hash: key, parent_key, diff_hash. Needed to identify the special columns for the loading procedure, depending on the table_stereotype.

field_name
Name of the field, containing the precalculated value in the source data set. Must only be set, when hash_name is not set.

stage_column_name
Name of the stage column, the hash can be taken from, when using stage table approach

data_mapping[]

(will be renamed to "field_mappings" in 0.6.3)

Json Path: $.parse_sets[].load_operations[]

List of the mappings of all fields to the table columns.

column_name
Name of the target column in the table

column_class
Datavault column class: business_key, content, content_untracked. Needed to identify the special columns for the loading procedure, depending on the table_stereotype.

field_name
Name of the field, the data must be taken from. Needed when using direct load from source to table or ods approach.

stage_column_name
Name of the stage column, the data must be taken from, when using the stage table approach.

update_on_every_load
When this property is set to true on at least one data mapping, the loading process must update the column in an existing current record if no new row is inserted. Identification of the record is done as follows: hub - hub key, link - link key, sat - parent key, multi active sat - <depends on multiactive historization pattern>, reference table- same set of values in all columns that are not udpated on every load (diff hash can be used as shortcut when available)

is_multi_active_key (optional)
Declares this column to be part of the multi-active key for a multi-active-satellite. Loading procedures, that support multi active historization loading patterns with a key, should use this as input.

This property is only set, when declared in the dvpd. There is no default value.

deletion_detection_rules[]

Json Path: $.parse_sets[].load_operations[]

Contains all deletion detection rules, that have to be applied to the table of this load operation.

procedure
declares deletion detection procedure for this rule. Suggested valid values are:

- "key_comparison" : Retrieve all (or a partition of) keys from the source, compare vault to the keys and create & stage deletion records for keys, that are not present anymore
- "deletion_event_transformation" : Convert explicit deletion event messages into a deletion record that is staged
- "stage_comparison" : The data retrieved and staged includes a complete set or partitions of the complete set. By comparing the whole vault against the stage, deletion records are created during the load from stage to the vault
- (more procedure names might be available in the actual load process implementation)

rule_comment (optional)
Name or short description of the rule. Enables more readable logging of exection progress and errors.
“All satellites of customer”

properties for other procedure

Please check out the definition in the dvpd specification for the following keywords

- key_fields[]
- partitioning_fields[]
- join_path[]
- active_keys_of_partition_sql

stage_columns[]

Json Path: $.parse_sets[]

The consolidated list of stage table columns.

stage_column_name
Name of the stage column. Must be used, when generating the table object.

column_type
Type of the stage column.Must be used, when generating the table object.

is_nullable
Boolean, when set to false, the column should have a NOT NULL constraint.

stage_column_class
Indicates, if the column contains metadata, hash values or data. Might be used to order the columns of the stage table by class, when creating the table.

hash_name (only for hashes)
Name of the hash in the hashes[] list. Should be used to retreive the hash recipe, needed to calculate the hash for this column.

field_name(only for data)
Name of the field in the fields[] list. Should be used to retreive the parsing parameters, needed to parse the field from the source data.

colunm_classes(only for data)
list of column classes of the target tables, this stage column is mapped to. Might be used to order the columns of the stage table by class, when creating the table. (e.g. business keys first, then content columns, then untracked columns)

License and Credits