DVPD Core Element Syntax Reference

DVPD core elements described here must be supported by any implementation in the same way. If an implementation leaves out elements for simplicity, it should implement a check and warning message, to prevent false assumptions.

The syntax must and can be extended by properties, needed for project specific solutions (e.g. data_extraction modules, data encryption frameworks). Documentation for these properties must be provided for every module in a separate document.

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

Root

dvpd_Version (mandatory)
Used to allow checking of compatibility. Must be set to the first version, that supports the used core elements. Minor version changes are kept backwardscompatible. Major version changes might modify structure, keywords and functionality.
"*1.0*"

pipeline_name (mandatory)
String to identify the pipeline. Must be unique over all pipelines managed by the system. Best practice for pipeline naming: Name of main the satellite/link, loaded by the pipeline.
"*rgopd_clicks_p1*"

pipeline_revision_tag (optional)
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.
"*1.1*" | "*x129sa8*"

pipeline_comment (optional)
String with human understandable information about the pipeline purpose.

record_source_name_expression (mandantory)
String used to build the record source name. Might contain placeholders in syntax {<name of placeholder}>, that will be replaced by specific string. Valid placeholders depend on the data_extraction and staging engine
"*sap.hr.depratments*"

data_extraction (mandatory)
Object, describing all necessary properties to access the data source
→ see “data_extraction”

model_profile_name (optional, default is “_default”)
Name of the model profile to be used. The model profile defines the names and types of data vault specific columns, declares the ruleset for hashing and more.

fields[](mandatory)
Array, describing every source field, its properties how to extract it from the source data format and how to map the data to the data vault model
→ see "fields"

data_vault_model[] (mandatory)
Array with objects, describing the data vault tables and relations
→ see "data_vault_modell"

deletion_detection (optional)
Object for declaring all necessary paramenter for deletion detection processes
→ see "deletion_detection"

data_extraction

subelement of root

fetch_module_name (mandatory)
name of the module, that is responsible to get the data from the source.This property would be used by a "master" process to determine, which module has to be chosen for loading the data. Available modules depend on the implementation and the variety of data transport procedures an source formats.
*"csv_file_ftp" | "xml_file_azure" | "json_stream_over_kafka"

>fetch module specific keys<
Depending on the used fetch module, there will be additional keys necessary to control the genereal fetching process. Valid keys must be documented in the module documentation. Please note, that field specific paramterers for fetching and parsing have to be placed as keys in the ***fields*** array

fields[]

subelement of root

field_name (mandatory)
Name of the field. Must be unique in the DVPD. Is normally used als the column name in the target table
"customer_id" | *"article _name"*

field_type (mandatory)
Expected type of the field. Delivered data must be compliant to this type, or will be rejected by the loading process. (Rejection method depends on the fetch module). Valid types depend on the fetch module, but should be chosen from SQL Syntax, so this technical type can be directly used in the definition of the data model.
"VARCHAR(200)" | *"INT8"* | *"TIMESTAMP"*

needs_encryption (optional boolean with default false)
When set to true, the data will be encrypted, according to the underlying concept for data protection (This is the only standardized core element regarding encryption. All other properties are defined by the specific method of encryption)

targets[] (mandatory)
Array, defining all taret tables, this field will be mapped to

→ targets

field_comment (optional)
Text that will be added as comment of the column in the stage data table and probably in generated documentation

> fetch module specific keys <
Depending of the data format and transport, there will be some more declarations necessary to identify the field in the source data. These properties can be added here. They must be documented in the fetch module documentation.

The order of elements in the array should be used for parsing positional data (csv, excel etc)..

targets[]

subelement of fields[]

This array must contain at least one target description. Fields, that are mapped to multiple tables, must have one entry for each target table

table_name (mandatory)
Name of the target table.(Must be defined in the data_vault_model section
"*rexmp_customer_hub*" | "*rgopd_ad_click_sat*"

target_column_name (optional)
Name of the column in the target table. If not defined, the field_name will be used
"*customer_number*"

target_column_type (optional)
Datatype of the target column in the database. Must be a valid Database type. If not defined, the technical_type of the field will be used
"*VARCHAR(200)*"

recursion_name (optional, only useful for business key fields, must be defined on a link, that is referring its hub as recursive_parent)
Declares this mapping to be used in a recursive reference. The name must be defined in a ***recursive_parent*** relation of a the link table.

field_groups[] (optional)
List of field groups this field mapping will be restricted to. If not defined, the mapping will be used in every field group.
"*[“Cust1”]*"

exclude_from_key_hash (optional,default=false,only useful for hub and link table mappings)
true = exclude the field from the calculation of the hub/link key. Used to define pure content column in the hub / link (this is rare but possible).

prio_in_key_hash (optional, default=0)
This property provides explicit control over the order of concatination of fields for the key hash calculation. It will overrule the implicit ordering, that is defeinde by the implementation. Implicit ordering will still be applied to columns of same prio.

exclude_from_change_detection (optional, default=false, only useful on mappings to historized satellites)
true = exclude the field from the change detection. Depending on the method of the target, this will modify the comparison SQL or the calculation of the diff hash.

prio_in_diff_hash (optional, default=0)
depending on the implementation the order of concatination of fields for the diff hash calculation is determined by the target_column name. This property provides more explicit control. Implicit ordering will still be applied to columns of same prio. When columns are added to productive satellites this will allow placing new columns behind already existing ones during concatination.

hash_cleansing_rules (optional)
object containing properties to describe a cleansing of the data before it is used in the hash.
→ see "hash_cleansing_rules"

column_content_comment (optional, default=comment of the field)
comment, that will be added to the column in the data vault model. Default it the comment of the field

data_vault_model[]

storage_component (optional)
Identification of the storage, this part of the model is placed. If not defined, there is only one storage component Valid valued depend on the processing modules and the overall architecture.
"*main_dwh_db*" | "*big_data_storage*"

schema_name (mandatory)
Name of the database schema, the tables are located. (this may also be the “database name”, since different database engines, adress this differently)

Especially for situations, where the schema name must also be used to provide dev/test/prod stages, it is recommended to declare parsable placeholders in the schema name. Those will be filled an runtime by the process, depending on the stage it runs in.
"*rvlt_accounting*"

tables[] (mandatory)
list of all tables, located in the declared schema

→ see tables

tables[]

table_name (mandatory)
Nname of the database table.
"*raccn_account_hub*"

stereotype (mandatory)
Data Vault Stereotype of the table. Valid values are: hub, lnk, sat, msat, esat
Depending on the stereotype, different properties have to be provided. The stereotype controls the processing for the load. The class of a column, generated for a mapped field is derived on the stereotype of the table as follows:

hub: mapped field is a business key except it is explicitly declared not to be (exclude_from_key_hash=true)

lnk: mapped field is a dependent child key except it is explicitly declared not to be (exclude_from_key_hash=true)

msat&sat: mapped field is part of the satellite

esat: there must not be any mapping of fields to an esat

table_content_comment (optional)
Comment to add to the table in the database

storage_component (optional, default=storage_component on schema level)
If the data vault tables are distributed over different storage engines (e.g. for keeping big data out of expensive database storage), this property can be used to identify the location. Valid values are depending on the implementation
"*fast_bi_db*" |"*big_data_storage_gcp*" |"*big_data_storage_hadoop*"

model_profile_name (optional, default is model_profile_name declared on pipeline level)
Name of the model profile to be used. The model profile defines the names and types of data vault specific columns, declares the ruleset for hashing and more. Declartion on table level allows interconnection between different profiles in the same model
"*postgres_main*"

"hub" specific properties

hub_key_column_name (mandatory)
Name of the hub key in the table
"*hk_raccn_account*"

"lnk" specific properties

Depending on mapped fields and the properties this can be a

- **normal link**
- **Link with dependend child keys**: Defined by mapping a field to the link table (without excluding it from the key)
- **recursive / hierarchical link**: is defined by recursive_parents declaration

link_key_column_name (mandatory)
Name of the link key in the table
"*lk_raccn_account_department*"

link_parent_tables[] (mandatory)
List of the table_names of all hubs, this link is connecting. The order of the tables in the list can be relevant to the hashing order of the link key (depends on processing engine and project conventions). In case, the processing engine enforces its own order, it should issue a warning, when the final order differs from the declarated. "*[“raccn_account_hub”, “raccn_department_hub”]*"

recursive_parents[] (optional)

List of recursive parent table declarations (e.g. for hierarchical links or “same as” links). Only tables that are link_parent_tables can be addressed here. The order of the tables in the list is relevant to the hashing order of the link key. In case, the processing engine enforces its own order, it should issue a warning, when the final order differs from the declarated. How recursive parent businesskeys are ordered in relation to link parent businesskey is a decision of the engine

→ recursive_parents

is_link_without_sat (optional)
must be set to true, to avoid warnings for links without an esat or sat.

tracked_field_groups[] (optional,only valid on links for is_link_without_sat=true)
list of field groups, this link will be processed for. The field groups must align with mappings of fields that are used as businesskey in the hubs, the link is referring

recursive_parents[]

table_name (mandatory, table must also be in link_parent_tables)
name of the link_parent_table, that is referenced again "*raccn_department_hub*"

recursion_name (mandatory) Name of the recursion. The name should be usable to extend the hub key column names, since in general the additional hub key columns in the the link will be generated by adding the recursion name. The name is referenced by the "recursion_name" property of a field, to declare the field to be used for this recursion relation.
"*master*"|"*parent*"|"*duplicate*"

field_group (mandatory)
⚠️ Might not be necessay ⚠️
field group defining fields for the business key columns of the hub, that have to be used for this relation
*"fg1,fg2"

“Satellite/multiactive satellite” specific properties

satellite_parent_table (mandatory)
Name of the hub/link table, this satellite is connected to "*raccn_account_hub*"

is_multiactive (optional, default=false)
when set to true, the declaration and processing for multiactive satellites will be applied (no primary key, awarenes of multiple active rows for change detection)

insert_changes_only (optional, default depends on model profile)
when set to true, incoming data is only added to the satellite if it differs from the latest version stored.

is_enddated (optional, default depends on model profile)
when set to true (default) meta data columns for historization enddating will be added to the table and loading process will execute enddating functions

uses_diff_hash (mandatory)
When set to true (default is defined in model profile), data change is detected by calculation of a hash value

ober all relevant columns and comparison of the hash value against the latest stored satellite row for every key.

diff_hash_column_name (might be ommitted, when the implementation is not using a diff hash)
Name of the colum that will contain the diff_hash
"rh_account_p1_sat"

has_deletion_flag (optional, default set by data model profile)
Determines if a deletion flag column will be added to the satellite.
Example:true

driving_keys[] (optional,must refer to a parent_key or dependent_child_key in the parent table of the satellite)
List of column names of the parent link, that are used as driving keys, to end former relations.

In general, the name must match the final name of the key column in the link. Especially in case of recursive relation, the method of creating the key name must be taken into account.
*"[“hk_raccn_account”]" | "[“hk_rerps_artice”,”year”,”month”]"

max_history_depth (optional)
depending on the implementation this will define a maximum depth of history in the satellite. Recommended thresholdtypes are: max_versions, max_valid_before_age

"Esat" specific properties

satellite_parent_table (mandatory, parent must be a link)
Name of the link table, this satellite is connected to
"raccn_account_department_lnk"

tracked_field_groups (optional)
list of field groups, this esat will be processed for. The field groups must align with mappings of field that are used as businesskey in the hubs, the link of the esat is referring

driving_keys[] (optional,must refer to a parent_key or dependent_child_key in the parent table of the satellite)
List of column names of the parent link, that are used as driving keys, to end former relations.

In general, the name must match the final name of the key column in the link. Especially in case of recursive relation, the method of creating the key name must be taken into account.
*"[“hk_raccn_account”]" | "[“hk_rerps_artice”,”year”,”month”]"

is_enddated (optional, default depends on model profile)
when set to true (default) meta data columns for historization enddating will be added to the table and loading process will execute enddating functions

max_history_depth (optional)
depending on the implementation this will define a maximum depth of history in the satellite. Recommended thresholdtypes are: max_versions, max_valid_before_age

"ref" specific properties

is_enddated (optional, default depends on model profile)
Defines, if the table will be historized by providing an enddate and using a diff hash

diff_hash_column_name (mandatory)
Colum that contains the diff_hash to determine the existence of the data constellation
"rh_country_iso_ref"

deletion_detection

subelement of root

Deletion detection can be implemented in multiple ways. DVPD will support declaration for some basic methods by using the following properties in the deletion_detection object.

procedure (mandatory)
provides the selection from different kind of procedures. 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 any more
- "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

key_fields[] (mandatory for "key_comparison", fields must be declared in fields[]. The fields must be mapped to businesskeys of a parent of the satellites)
Names of the fields, used to retrieve the list of still available keys in the source. The modell mapping provides the necessary join relation to the satellite tables for determening the currently valid values in the vault.

deletion_rules[] (mandatory)
List of deletion rules. The order of the the rules in this array must be obeyed.

→ deletion_rules[]

> procedure specific properties < For other procedures, then the defined above there might be other properties necessary.

deletion_rules[]

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

satellites_to_delete[] (mandatory,only declared satellite table names allowed)
List of satellite table names, on which to apply the deletion detection rule. The satellites must share the same parent
"rsfdl_cusmomer_p1_sat","rsfdl_customer_p2_sat"

partitioning_columns[] (optional,only declared field names are allowed)
List of Columns (and therefore vault columns), that define the range of data where stage has a complete set of rows (this can be content or even table keys). Only active satellite rows that are related to the staged values in these fields, will be checked for deletion. If this property is not set, a complete dataset is assumed to be in the

stage table.
"market_id"

join_path[] (optional,must contain all tables need to be joined to reach the partitioning columns)
Describes the join path in the model to get from the tables to delete to the tables with partitioning fields. The path begins with the parent table of all listed satellites to delete. The path must not branch except when adding satellites to provide partitioning columns or restrict the validity of links. The path can skip unnecessary tables (e.g. hubs, where businesskey is no partition criteria).

Example:

```
Model: contract_p1_sat + contract_p2_sat -> contract_hub(contId)
            <-customer_contract_lnk/esat->
            customer_hub(country,custId)

satellite_tables: \[contract_p1_sat,contract_p2_sat]

partitioning_fields: \[country]

join_path: \[customer_contract_lnk,customer_contract_esat,customer_hub]

This will delete all acitve rows from contract_from_customer_p1_sat and
contract_from_customer_p2_sat where the country of the customer is in stage but
not the contId
(=Hub key of satellite = second hub key in link)
```

active_keys_of_partition_sql (optional, used for cases, that can't be described by partitioning_fields and joins_path, only appliable for a single sattellite to delete)
To solve more complex scenarios for deletion detection, a Select statement can be provided, that determines all active keys of a satellite for a specific partition. The engine will only compare the given set with the staged data and insert deletion records accordingly . (If by ELT or ETL depends on the engine) "join_path" and "partitioning_columns" must be empty.

> procedure specific properties < For other procedures, then the defined above there might be other properties to be declared in the deletion_rule.

Open concepts

There are some concepts open to be defined later. These will result in some upcoming syntax elements:

Hash assembly rules

"lnk" specific properties/ link_key_assemble_rule (drafted option, default: "p/p-r/p-d/ea")
Defines the rule, how to order the businesskeys for the link key concatenatenation. See section "Hash concat order rule declaration" for detailed specification

"lnk" specific properties/link_key_explicit_content_order[] (optional, must contain all relevant columns of parents and link
Specifies directly the order of businesskeys and dependent child key columns for hashing. Disables a link_key_assemble_rule

content_enddate_columns[] (optional)
List of column pairs, where an addtional enddate processing should be applied (might be interesting, when modification dates of the source are relevant for queries)

Hash concat order rule declaration

For hub keys and diff hashes

The order of the fields for concantenation during hashing can completly be controlled by using the prio_in_hash_key and prio_in_diff hash declaration in the target section.

Whithout declaration the order will be alphabetical with the target column name.

The priority attirbutes have a higher siginificance then the column name. So defining a priority on every mapping gives full control.

For Link keys

This is only a draft and not implemented yet

Participation of fields in link keys is derived from the data model relations. An explicit declaration of the order is not yet possible, but will be added for edge cases later.

Until then, the main control over the order of hashing in the link key is by applying ordering rules.

There are various possibilities how to order the businesskeys of the multiple hubs / hub relations before concantenating them for hashing. Even though, not all execution engines might support the whole bandwidth, it should be defined in a standardizes way. Engines should reject unsupported patterns.

Since the hash rules are an essential part of the interface, the following declaration standard is defined

```
syntax: [Group1] -> [group 2] ->…

One Group consists of :  <source elements class>/<ordering rules>

Source elements classes are:
        “p” - (normal) parent tables
        “r” - recursive parent tables
        “d” - dependent child key elements

The sequence of source elements in the same group is not relevant

Ordering rules are:
        “a” alphabethical target columnname
        “e” explicit priorities
        “p” same order as in the parent table
        “o” order in declaration array(only possible for separated “p” and “r”
source)
        “t” alphabetical target table name

The sequence of Ordering rules in the same group define the hierarchy of criteria
```

Examples:

- "p:op -> r:op -> d:ea"
  - "p:op" all bk of parent tables, taking first all bk first from first table in the "parent_tables" array and so on. Sorting the bk of the same table like in the parent table
  - "r:op" all bk of recursive parent tables, taking first all bk first from first table in the "recursive_tables" array and so on. Sorting the bk of the same table like in the parent table
  - "d:ea" all dependent child key fields, ordered by priority and name
- "prd/ta"
  - Order fields by their source table name and field name. Dependent child key fields have the link tables itseld as table name

Licence and Credits