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. Properties and value, not defined in the core, must be prefixed with a "x", followed by an abbreviation of the using module. (e.g. "xenc" for encryption module of cimt)

For examples please look in : Data Vault method coverage and syntax examples

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

Root

dvpd_version (mandatory) *defines: compiler feature control*
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.
Examples: "*1.0.0*"

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

pipeline_revision_tag (optional) *defines: Build pipeline control, auditing*
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.
Examples: "*1.1*" | "*x129sa8*"

pipeline_comment (optional) *defines: documentation*
String with human understandable information about the pipeline purpose.

record_source_name_expression (mandantory) *defines: meta data content*
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”) *defines: general declarations*
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_model”

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

stage_properties[] (mandatory)
Object with stage table declarations. (one for every storage component)
→ see "stage_properties”

data_extraction

json path: /

fetch_module_name (mandatory) *defines: fetch behaviour*
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 and source formats.
*"csv_file_ftp" | "xml_file_azure" | "json_stream_over_kafka"

>fetch module specific keys< *defines: fetch behaviour* 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[]

json path: /

field_name (mandatory) *defines: source parsing, model structure*
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) *puprpose: source parsing, model structure*
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*"

field_value (optional) *defines: data generation*
>will be implemented in later version<
Sets a value for the field, that is constant over the whole processing. This field will not be parsed from the source. Depending on the consuming code generator, the value might be an expression, interpreted and replaced with an actual value by the final loading process (e.g. for inserting the load date or some other information from the processing environment or metadata of the processed work item). Transforming incoming data should be expressed in other properties.

It is recommended to use the "${value name}" syntax for this purpose (e.g. "${CURRENT_DATE}").

The following constants are defined by the core syntax

constant	content
${NULL}	Null value
${EMPTY}	Emtpy String value
${CURRENT_DATE}	Date of the day of the loading process
${CURRENT_TIMESTAMP}	Timestamp of the loading process
${CURRENT_TIME}	Time of the loading process
${RELATION_NAME}	Name of the processed relation

needs_encryption (optional boolean with default false) *defines: encryption processing*
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 target tables, this field will be mapped to
→ targets

field_comment (optional) *defines: documentation, table DDL*
Text that will be added as comment of the column and probably in generated documentation

> fetch module specific keys < *defines: parsing*
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[]

json path: /fields[]/

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

table_name (mandatory)*defines: mapping*
Name of the data vault table. (Must be defined in the data_vault_model section)
"*rexmp_customer_hub*" | "*rgopd_ad_click_sat*"

column_name (optional)*defines: model structure*
Name of the column in the data vault table. If not defined, the field_name will be used.
"*customer_number*"

column_type (optional)*defines: model structure*
Datatype of the column in the data vault table. Must be a valid type of the platform database. If not defined, the technical_type of the field will be used
"*VARCHAR(200)*"

prio_for_column_position (optional, default is 50000)*defines: table DDL*
>will be implemented later<
Defines the position of the column in the table declaration. Columns of the same prio will be ordered alphabetically by the column name.

prio_for_row_order (optional, default is 50000) *defines: diff hash input assembly*
>will be implemented later<
Defines the position of the column, when ordering rows for calculation of the group hash for multiactive satellite loading. Columns of the same prio will be ordered alphabetically by the column name. The high default value sets all columns without any declaration at the end of the list.

row_order_direction (optional, default=ASC) *defines: diff hash input assembly*
will be implemented later
Defines the direction of the order of content of this column, for calculation of the group hash for multiactive satellite loading. Possible values are "ASC" and "DESC".

relation_names[] (optional) *defines: mapping, data model, load operations*
Declares this mapping to be used only in specific relations. It should be valid as a SQL name, since it will be used as name extension for staging columns.

The name must be a valid ***relation_name*** depending on the role of the field.

- Business key - The name defines a valid relation, the targeted hub is supporting
- Dependent child key - The name must match a relation name, that is supported by a parent of the link
- Data excluded from key for hub or link - The name must match a relation name, supported by the target
- Satellite Content - The name must match a relation name, supported by the parent

To explicitly declare participation in the main (unnamend) relation use "/" as name. This is the only way to declare the participation in a subset of relations that contains the unnamed relation.

When ommitting this property, the field mapping participates in relations as follows:

- when the field is the only field mapped to a target column, the mapping is used in all relations
- when there are multiple fields mapped to a target column, the mapping is used only in the main (unnamed) relation.

["parent"] , ["child1","child2"], ["/","Sibling"]

exclude_from_key_hash
(optional, default=false, only useful for hub and link table mappings)
defines: key hash assembly, business key identification
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) *defines: key hash assembly*
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 defined by the implementation. Implicit ordering will still be applied to columns of the same prio.

exclude_from_change_detection (optional, default=false, only useful on mappings to historized satellites)
defines: satellite load, diff hash assembly
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) *defines: diff hash assembly*
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 the 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) *defines: hash assembly*
object containing properties to describe a cleansing of the data before it is used in the hash.
→ see "hash_cleansing_rules"

update_on_every_load (optional, default=false, can't be set for business keys) *defines: load steps*
announced for upcoming version
if not supported on specific stereotypes, this must throw a warning
Forces the load process to update the column with the staged value every time (e.g. for a last seen date in a hub). When used in satellite or reference table mappings, only the current row for the parent key should be updated.

column_content_comment (optional, default=comment of the field) *defines: documentation, table DDL*
comment that will be added to the column in the data vault model. Default is the comment of the field

data_vault_model[]

Json Path: /

storage_component (optional) *defines: load procedure, SQL dialect*
Identification of the storage, this part of the model resides. If not defined, there is only one storage component. Valid values depend on the processing modules and the overall architecture.
"main_dwh_db" | "big_data_storage"

schema_name (mandatory) *defines: database structure*
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. It is recommended to use the "${value name}" syntax for this purpose (e.g. "${STAGE_TAG}"). Those will be filled at 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[]

Json Path : /data_vault_mode[]/

table_name (mandatory) *defines: model structure*
Name of the database table.
"*raccn_account_hub*"

table_stereotype (mandatory) *defines: model structure, loading procedure*
Data Vault Stereotype of the table. Valid values are: hub, lnk, sat, ref
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 from the stereotype of the table as follows:

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

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

sat: mapped field is part of the satellite

ref: mapped field is part of the reference table

Satellites without any mapped content column are allowed (effectivity satellites).

table_content_comment (optional) *defines: documentation, table ddl*
Comment to add to the table in the database

storage_component (optional, default=storage_component on schema level) *defines: load procedure, SQL dialect*
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 depend 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) *defines: general declarations*
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

Json Path : /data_vault_mode[]/tables[]

hub_key_column_name (mandatory)*defines: model structure*
Name of the hub key in the table. (Currently this name must be unique over all tables in the declared model. Future versions will extend the syntax to allow the same name in different tables, even though it is highly recommended to have unique hub key names)
"hk_raccn_account"

"lnk" specific properties

Json Path : /data_vault_mode[]/tables[]

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)

link_key_column_name (mandatory)*defines: model structure*
Name of the link key in the table
"lk_raccn_account_department"

link_parent_tables[] (mandatory)*defines: model structure*
The following two options are possible contents

1. Just a list of the table_names of all hubs, this link is connecting.
2. A list of json objects with full link parent property declarations

→ link_parent_tables[]

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 one declared in the DVPD.
Example for 1: "[“raccn_account_hub”, “raccn_department_hub”]"
Example for 2: "[{"table_name":“raccn_account_hub”, "relation_name":"PARTNER"}, {"table_name":“raccn_account_hub”}]"

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

link_parent_tables[]

Json Path : /data_vault_mode[]/tables[]

By using the full property declaration syntax for parent tables, multiple relations to the same hub can be expressed properly

table_name (mandatory)*defines: model structure*
name of the link_parent_table
"raccn_department_hub"

relation_name (mandatory for every table, declared more then once in the list) *defines: model structure, loading operations*
Name of the relation. The name should be usable to extend the hub key column names. The name is referenced by the "relation_name" property of a field, to declare the field to be used for this relation.
"master"|"parent"|"duplicate"

hub_key_column_name_in_link (optional)*defines: db element naming*
Name of the hub key columns, to be used for the mapping of this relation. If ommitted the hub key column name will be generated by adding the relation name to the original hub key column name

"sat" specific properties

Json Path : /data_vault_mode[]/tables[]

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

is_multiactive (optional, default=false) *defines: loading procedure, diff hash assembly*
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)

compare_criteria (optional, default depends on model profile)*defines: loading procedure*
defines the criteria that have to be met, for inserting from stage into the satellite. Valid settings are:

- 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 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 (preventing duplication by repeated loads must be solved by load orchestration)

The settings "key", "current" and "key+current" should be supported by every implementation, since they belong to the core of data vault. The settings "key" and "none" might remove a declared diff hash column, or at least will leave out the check for a diff_hash even, when uses_diff_hash is true.

The settings "key" and "none" make the setting of diff_hash_column_name optional

is_enddated (optional, default depends on model profile)*defines: loading procedure, table structure*
when set to true (default), metadata columns for historization enddating will be added to the table and the

loading process will execute enddating functions

uses_diff_hash (optional, default depends on model profile)*defines: loading procedure, table structure*
When set to true, data change is detected by calculation of a hash value over all relevant columns and comparison of the hash value.

diff_hash_column_name (depending on uses_diff_hash setting and compare_criteria) *defines: db element naming*
Name of the column that will contain the diff_hash. (Currently, this name must be unique in the declared model, since all diff hash columns will be part of the stage table. Future versions will extend the syntax to allow the same physical name in different tables of pipeline model)

This must be set, when uses_diff_hash is true and compare_criteria is not "key" or "none". The diff hash will be put in the table for any compare_criteria setting, when declared.
"rh_account_p1_sat"

has_deletion_flag (optional, default set by data model profile)*defines: loading procedure, table structure*
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 link table of the satellite) *defines: loading procedure*
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 a hub key column in the link. Especially in case of multiple relations to the same table, the method of creating the key column name must be taken into account.
*"["hk_raccn_account"]" | "["hk_rerps_artice","year","month"]"

tracked_relation_name (optional, only valid on effectivity satellites) *defines: loading operations*
Name of the relation this satellite will track the validity for. This is only used for satellites whithout any field mapping. The relation name must be valid for the satellites parent.

history_depth_criteria (mandatory when history_depth_limit is set) *defines: loading procedure*
>announced for upcoming version<
Defines the criteria to determine the history depth

- versions : the number of versions for every key is limited to the given threshold (i.e. only store the last x entries for a given key).
- enddate_days : the number of days the enddate is behind the current day

history_depth_limit (optional) *defines: loading procedure*
>announced for upcoming version<
defines a maximum depth of history in the satellite. No declaration or nagative values are treated as "no limit". When the satellite is loaded, all "not current" rows, that are beyond the given threshhold, are deleted.

"ref" specific properties

Json Path : /data_vault_mode[]/tables[]

is_enddated (optional, default depends on model profile) *defines: loading procedure, table structure*
Defines, if the reference table will be historized by providing an enddate

uses_diff_hash (optional, default depends on model profile) *defines: loading procedure, table structure*
When set to true, existence of a specific value combination in the source is detected by calculation of a hash value over all relevant columns and comparison of the hash value against the actual valid rows in the reference table.

diff_hash_column_name (mandatory) *defines: db element naming*
Colum that contains the diff_hash to determine the existence of the data constellation
"rh_country_iso_ref"

history_depth_limit (optional) *defines: loading procedure*
>announced for upcoming version<
defines a maximum depth of history in the reference table in days . No declaration or nagative values are treated as "no limit". When the table is loaded, all rows, that are beyond the given threshhold, are deleted. (in reference tables, the row can only age "on their own", they have no "key" to measure a version count)

deletion_detection

Json Path: /

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) *defines: loading procedure*
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 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

key_fields[] (mandatory for "key_comparison", valid fields must be declared in fields[]. The fields must be mapped to business keys of a parent of the satellite) *defines: loading procedure*
Names of the fields, used to retrieve the list of still available keys in the source. The model mapping provides the necessary join relation to the satellite tables, for determining the currently valid values in the vault.

partitioning_fields[] (optional) *defines: loading procedure*
Names of the fields, which identify fully delivered datasets (partitions) in the current load. If set, the deletion detection wil be restricted to these partitions.

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 there might be other properties necessary.

deletion_rules[]

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

tables_to_cleanup[] (mandatory, only declared table names allowed) *defines: loading procedure*
List of table names, on which to apply the deletion detection rule. Multiple entries are only allowed for satellites of the same parent.
"rsfdl_customer_p1_sat","rsfdl_customer_p2_sat"

join_path[] (optional, must contain all tables needed to be joined to reach the partitioning columns) *defines: loading procedure*
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, whose business keys are not a partition criteria).

An empty join path means that the partitioning columns are all in the table to cleanup. It can't be set, when there are no partitioning_fields.

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 is present in the stage but
combination of country and contId is missing
    (=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 applicable for a single satellite to delete) *defines: loading procedure*
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) The DVPD properties "join_path" and "partitioning_columns" must be empty.

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

stage_properties[]

Json Path: /

Contains the declaration of the stage table locations. In common scenarios there will be only one. In case of a distributed model using ELT from stage to vault, the stage table can be placed on every target.

storage_component (optional) *defines: load procedure, SQL dialect*
Identification of the storage, this staging table is placed. If not defined, there is only one storage component. Valid values depend on the processing modules and the overall architecture.
"main_dwh_db" | "big_data_storage"

stage_schema (mandatory) *defines: database structure*
Name of the schema the stage table is placed in.
*"stage_rvlt"

stage_table_name (optional) *defines: db element name*
Name of the stage table. Default is the name of the pipeline.
*"srvlt_crm_person_p1"

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 business keys for the link key concatenation. 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
Directly specifies the order of business keys 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 attributes have a higher siginificance than 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 business keys of the multiple hubs / hub relations before concatenating them for hashing. Even though not all execution engines might support the whole bandwidth, it should be defined in a standardized 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” - 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 column name
        “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 defines the hierarchy of criteria
```

Examples:

- "p: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
  - "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 itself as table name

License and Credits