VDP Naming Conventions

Revision 20240229

Denodo Inc.: 525 University Avenue, Suite 31, Palo Alto, CA 94301. USA www.denodo.com

CONTENTS

1 GOAL....3

2 GENERAL RECOMMENDATIONS....4

3 VDP PROJECT STRUCTURE....4
3.1 PROJECT LAYERS....4
3.2 FOLDER STRUCTURE....4

4 VDP NAMING CONVENTIONS....7
4.1 DATA SOURCES....7
4.2 BASE VIEWS....7
4.3 INTEGRATION VIEWS....7
4.4 INTERFACES....8
4.5 FINAL VIEWS....8
4.6 STORED PROCEDURES....9
4.7 ASSOCIATIONS....9
4.8 WEB SERVICES/WIDGETS....9
4.9 JMS LISTENERS....10
4.10 SUMMARIES....10
4.11 VIEW ATTRIBUTES....11

1 GOAL

This document enumerates the proposed naming conventions for all the elements that can be created as part of a Virtual DataPort database.

2 GENERAL RECOMMENDATIONS

- It is strongly recommended for organizations to define a naming conventions document to simplify the reading and understanding of elements created in Virtual DataPort.
- Use English.
  - If the final consuming applications require a different language, use that language for the final views and/or stored procedures that are directly accessed by these applications.
- Name views and attributes in lowercase, and separate words by an underscore (“_”).
- Name views in their singular form (customer, contract, etc.).
- Use the `Metadata > Description` option to document the elements in a database.
- Use the attribute’s `description` field to document every attribute.
- Do not include as part of the name anything that depends on the environment where the element is being created.

3 VDP PROJECT STRUCTURE

3.1 PROJECT LAYERS

A data virtualization project has several different layers:

- **Connectivity**: Related to the physical systems. Data sources and base views are part of this layer.
- **Integration**: Includes the combinations and transformations for the next layers. No directly consumed views at this level.
- **Business Entities:** Canonical business entities exposed to all users.
- **Report Views:** Pre-built reports and analysis frequently consumed by users.
- **Data Services:** Web services for publishing views from other levels. Can contain views needed for data formatting and manipulation.

3.2 FOLDER STRUCTURE

Having an organized folder structure is critical, especially for complex projects, as it helps in locating elements and clearly understanding the different elements of a project.

Based on the project structure described above, these are the recommendations for folders:

- Have one folder per layer.

- Create additional folders for other elements (e.g. associations and JMS listeners.).
- In complex projects, additional subfolders can be added to group related elements.

This is an example of the recommended root folders:

1 - Connectivity

This folder stores the data sources and base views of the project. Additional subfolders can be added at this level:

- 1 - data sources
- 2 - base views

2 - Integration

The integration views created for combining the different base views are stored in this folder. Additional subfolders can be created for grouping the related integration views.

3 - Business Entities

The interfaces representing the business entities will be stored at this level. Additional subfolders can be added. The implementation views, since they are not going to be directly accessed by the client application, can remain at the integration level.

4 - Report Views

Interfaces representing the pre-built reports published to the final application will be stored in this folder.

5 - Data Services

This folder stores data services:

- Web services:
  - SOAP
  - REST
- Widgets

6 - Associations

This folder stores associations.

7 - JMS Listeners

This folder stores JMS listeners.

8 - Stored Procedures

This folder stores custom stored procedures developed using the Virtual DataPort API.

9 - Summaries

This folder stores summaries.

Below, you can see some sample VQL that can be execute in a new database to create the recommended folder structure:

```
CREATE OR REPLACE FOLDER '/01 - Connectivity' DESCRIPTION 'This folder
stores the data sources and base views of the project' ;

CREATE OR REPLACE FOLDER '/01 - Connectivity/01 - Data Sources' DESCRIPTION
'This folder stores the data sources of the project' ;

CREATE OR REPLACE FOLDER '/01 - Connectivity/02 - Base Views' DESCRIPTION
'This folder stores the base views of the project' ;

CREATE OR REPLACE FOLDER '/02 - Integration' DESCRIPTION 'The integration
views created for combining the different base views are stored in this
folder' ;

CREATE OR REPLACE FOLDER '/03 - Business Entities' DESCRIPTION 'The
interfaces representing the business entities will be stored at this level.
Additional subfolders can be added. The implementation views, since they are
not going to be directly accessed by the client application, can remain in
the integration level.' ;

CREATE OR REPLACE FOLDER '/04 - Report Views' DESCRIPTION 'Interfaces
representing the pre-built reports published to final application will be
stored in this folder.' ;

CREATE OR REPLACE FOLDER '/05 - Data Services' DESCRIPTION 'This folder
stores data services.' ;

CREATE OR REPLACE FOLDER '/06 - Associations' DESCRIPTION 'This folder
stores associations.' ;

CREATE OR REPLACE FOLDER '/07 - JMS Listeners' DESCRIPTION 'This folder
stores JMS listeners' ;

CREATE OR REPLACE FOLDER '/08 - Stored Procedures' DESCRIPTION 'This folder
stores Stored procedures' ;

CREATE OR REPLACE FOLDER '/09 - Summaries' DESCRIPTION 'This folder stores
Summaries' ;
```

4 VDP NAMING CONVENTIONS

4.1 DATA SOURCES

Data sources can be named as follows:

- **ds_{data source name}**
  - E.g.:
    - ds_**internet**
    - ds_**soap_billing**
    - ds_**phone**

4.2 BASE VIEWS

Base views can be named as follows:

- **bv_{data source name}_{source table / description (if sql-query view) / source ws operation / stored procedure / ... (optional)}**
  - **{data source name}** is the name of the data source over which the base view is created.
    - E.g.:
      - bv_**internet**_incident
      - bv_**phone**_incident
  - **{source table / description (if sql-query view) / source ws operation / stored procedure / ...}** is the name of the table, stored procedure, web service operation, or any descriptive text that identifies what it is imported from the data source. It is optional if only one base view is created for a data source. For example, to import a single XML file called “user_incidents.xml” in Virtual DataPort, the data source will be called “ds_user_incident” and the base view will be called “bv_user_incident”.
    - E.g.:
      - bv_internet_**incident**

Using this naming convention allows for quickly listing all the views related to a specific data source when searching by the data source name.

With this approach, all the base views from a specific data source will also appear together if using external JDBC/ODBC clients to access a Virtual DataPort database.

4.3 INTEGRATION VIEWS

As defined in the project structure section, integration views are those derived views created for combination and transformation but should not be directly accessed by final client applications. These integration views can be named as follows:

- **iv_{integration view descriptive name}_{order #}_{action, description} (optional)**
    - **{integration view descriptive name}** is the descriptive name for this integration view.
        - In order to get all related integration views ordered, it is recommended to use the same name for all of them.
        - This descriptive name can be:
            - Similar to the base view used.
            - Similar to the final view that will use this integration view.
        - E.g.:
            - `iv_`**`data_customer_mail`**`_03_flatten`
            - `iv_`**`data_personal`**`_05_register_address`
    - **{order #}** is the numeric value used to order the views.
        - If an integration view is created from different integration views, the new integration view should use the number following the highest number of the views that the new view depends on.
        - E.g.:
            - `iv_data_customer_mail_`**`03`**`_flatten`
            - `iv_data_personal_`**`05`**`_register_address`
    - **{action, description}** is a descriptive text about the action or meaning that identifies what the integration view does.
        - E.g.:
            - `iv_data_customer_mail_03_`**`flatten`**
            - `iv_data_07_`**`union_customer_info`**

4.4 INTERFACES

Interfaces should have a descriptive name that makes them easy to identify for final users:

- **{interface name}**
    - E.g.:
        - **`customer`**
        - **`customer_address`**
        - **`customer_email`**

4.5 FINAL VIEWS

Final views contain information that is directly accessed by external systems and/or published for their consumption. Final views can be named as follows:

- **{identifier}_{by_input_parameters (optional)}**
    - **{identifier}** is the identifier for this final view.
        - E.g.:
            - **`customer`**
            - **`get_customer_info`**

- **{by_input_parameters}** is used only if this view has mandatory input parameters.
  - Specify the input parameters separated by "_" in the view name.
  - However, if the final view has many input parameters, this notation should not be used.
  - E.g.:
    - `customer_search_`**`by_name_lastname`**

4.6 STORED PROCEDURES

Stored procedures can be named as follows:

- **sp_{sp_name}_{by_input_parameters (optional)}**
  - **{sp_name}** is the descriptive name of the created stored procedure.
    - E.g.:
      - `sp_`**`customer_global_search`**
  - **{by_input_parameters}** is used only if the stored procedure is always executed with input parameters.
    - Specify the input parameters separated by "_" in the name of the stored procedure.
    - If the final view has a lot of input parameters, this notation should not be used.
    - E.g.:
      - `sp_customer_search_`**`by_name_lastname`**

4.7 ASSOCIATIONS

Associations can be named as follows:

- **a_{principal entity name}_{related entity}**
  - E.g.:
    - a_**`customer_contract`**

4.8 WEB SERVICES/WIDGETS

The web service or widget name should be a business noun, business concept, or business process.

The naming of these elements depends on each customer and project naming conventions.

The web service operation names should be similar to the final view that implements the operation.

These are the naming conventions for published Denodo web services:

- **SOAP**:
  - **{web service name}:**
    - The name of the SOAP web service:
      - Should include the “service” suffix.
      - Should be in lowercase.
      - Should not use separators (“ ”, “_”, etc.).
      - Should be singular.
      - Should not use verbs.
    - E.g.:
      - **internetincident**service
      - **userinvoicing**service
  - **{Operations}**:
    - Name of the SOAP web service operations:
      - Should not include separators (“ ”, “_”, etc.).
      - Should use the name of the view.
      - Should use verbs.
    - E.g.:
      - **getINTERNETINCIDENTSBySTATUS**
      - **insertINTERNETINCIDENTS**
- **REST**:
  - **{web service name}:**
    - Name of the REST web service:
      - Should not include the “service” suffix.
      - Should be in lowercase.
      - Can use the “_” separator.
      - Should be singular.
      - Should not use verbs.
    - E.g.:
      - **internet_incident**
      - **sale**

4.9 JMS LISTENERS

JMS listeners can be named as follows:

- **jms_{jms name}**
  - E.g.:
    - jms_**customer**

4.10 SUMMARIES

Summaries can be named as follows:

- **s_{summary name}**
  - E.g.:
    - **s_total_by_year_store**

4.11 VIEW ATTRIBUTES

- Name identifier fields (id, code, etc.) as `viewname_id`.
  - Example: If the view name is `bv_customer_mail` and it has an `id` field, it should be named `customer_mail_id`.
- Name fields from other views as `relatedviewname_id`.
- Name input parameters as `input_descriptive_name`.