The following part will be added to the Section 6 of the SDMX Standards ("SDMX Technical Notes") as the last Section (n.10), before the Annex

10 Validation and Transformation Language (VTL)

10.1 Introduction

The Validation and Transformation Language (VTL) supports the definition of Transformations, which are algorithms to calculate new data starting from already existing ones[1].

The purpose of the VTL in the SDMX context is to enable the:

- definition of validation and transformation algorithms, in order to specify how to calculate new data from existing ones;
- exchange of the definition of VTL algorithms, also together the definition of the data structures of the involved data (for example, exchange the data structures of a reporting framework together with the validation rules to be applied, exchange the input and output data structures of a calculation task together with the VTL transformations describing the calculation algorithms);
- compilation and execution of VTL algorithms, either interpreting the VTL transformations or translating them in whatever other computer language is deemed as appropriate.

It is important to note that the VTL has its own information model (IM), derived from the Generic Statistical Information Model (GSIM) and described in the VTL User Guide. The VTL IM is designed to be compatible with more standards, like SDMX, DDI (Data Documentation Initiative) and GSIM (Generic Statistical Information Model), and includes the model artefacts that can be manipulated (inputs and/or outputs of transformations) and the model artefacts that allow the definition of the transformation algorithms.

The VTL language can be applied to SDMX artefacts by mapping the SDMX IM model artefacts to the model artefacts that VTL can manipulate. Thus, the SDMX artefacts can be used in VTL as inputs and/or outputs of transformations. It is important to be aware that the artefacts do not always have the same names in the SDMX and VTL IMs, nor do they always have the same meaning. The more evident example is given by the SDMX “dataset” and the VTL “dataset”, which do not correspond one another: as a matter of fact, the VTL “dataset” maps to the SDMX “dataflow”, while the SDMX “dataset” has no explicit mapping to VTL (such an abstraction is not needed in the definition of VTL transformations). A SDMX “dataset”, however, is an instance of a SDMX “dataflow” and can be the artefact on which the VTL transformations are executed (i.e., the transformations are defined on “dataflows” and are applied to dataflow instances, that can be SDMX datasets).

The VTL expressions are accessed through the maintainable artefact “Transformation Scheme” which is composed of “Transformation” nameable artefacts. Each Transformation contains a VTL expression.

This section does not explain the VTL language or any of the content published in the VTL guides. Rather, this is a description of how the VTL can be used in the SDMX context and applied to SDMX artefacts.

---

[1] The Validation and Transformation Language is a standard language designed and published under the SDMX initiative. VTL is described in the VTL User and Reference Guides available on the SDMX website https://sdmx.org.

10.2 References to SDMX artefacts from VTL statements

10.2.1 Introduction

The VTL transformations can manipulate SDMX artefacts (or objects) by referencing them through pre-defined conventional names (aliases).

The alias of a SDMX artefact can be its URN (Universal Resource Name), an abbreviation of its URN or another user-defined name.

In any case, the aliases used in the VTL transformations have to mapped to the SDMX artefacts through the `VtlMappingScheme` and `VtlMapping` classes (see the section of the SDMX IM relevant to the VTL).

A `VtlMapping` allow specifying the aliases to be used in the VTL expressions to reference SDMX artefacts. The `VtlMappingScheme` is a container for zero or more `VtlMapping`. The correspondence between an alias and a SDMX artefact must be one-to-one, meaning that a generic alias identifies one and just one SDMX artefact while a SDMX artefact is identified by one and just one alias. In other words, within a `VtlMappingScheme` an artefact can have just one alias and different artefacts cannot have the same alias.

The references through the URN and the abbreviated URN are described in the following paragraphs.

10.2.2 References through the URN

The SDMX URN[2] is the concatenation of the following parts, separated by special symbols like dot, equal, asterisk, comma, and parenthesis:

- SDMXprefix
- SDMX-IM-package-name
- class-name
- agency-id
- maintainedobject-id
- maintainedobject-version
- container-object-id [3]
- object-id

The generic structure of the URN is the following:

SDMXprefix**.**SDMX-IM-package-name**.**class-name**=**agency-id**:**maintainedobject-id **(**maintainedobject-version**).***container-object-id**.**object-id

[2] For a complete description of the structure of the URN see the SDMX 2.1 Standards - Section 5 - Registry Specifications, paragraph 6.2.2 ("Universal Resource Name (URN)").

[3] The container-object-id can repeat and may not be present

The **SDMX prefix** is “urn:sdmx:org”, always the same for all SDMX artefacts.

The **SDMX-IM-package-name** is the concatenation of the string “sdmx.infomodel.” with the package-name which the artefact belongs to. For example, for referencing a dataflow the SDMX-IM-package-name is “sdmx.infomodel.datastructure”, because the class “Dataflow” belongs to the package “datastructure”.

The **class-name** is the name of the SDMX object class which the SDMX object belongs to (e.g., for referencing a dataflow the class-name is “Dataflow”). The VTL can reference SDMX artefacts that belong to the classes `dataflow`, `dimension`, `measureDimension`, `timeDimension`, `primaryMeasure`, `dataAttribute`, `concept`, `conceptScheme`, `codelist`.

The **agency-id** is the acronym of the agency that owns the definition of the artefact, for example for the Eurostat artefacts the agency-id is “ESTAT”). The agency-id can be composite (for example AgencyA.Dept1.Unit2).

The **maintainedobject-id** is the name of the maintained object which the artefact belongs to, and in case the artefact itself is maintainable[4], coincides with the name of the artefact. Therefore the maintainedobject-id depends on the class of the artefact:

- if the artefact is a `dataflow`, which is a maintainable class, the maintainedobject-id is the `dataflow` name (dataflow-id);
- if the artefact is a `dimension`, `measureDimension`, `timeDimension`, `primaryMeasure` or `dataAttribute`, which are not maintainable and belong to the `dataStructure` maintainable class, the maintainedobject-id is the name of the `dataStructure` (dataStructure-id) which the artefact belongs to;
- if the artefact is a `concept`, which is not maintainable and belongs to the `conceptScheme` maintainable class, the maintainedobject-id is the name of the `conceptScheme` (conceptScheme-id) which the artefact belongs to;
- if the artefact is a `conceptScheme`, which is a maintainable class, the maintainedobject-id is the name of the `conceptScheme` (conceptScheme-id);
- if the artefact is a `codelist`, which is a maintainable class, the maintainedobject-id is the `codelist` name (codelist-id).

The **maintainedobject-version** is the version of the maintained object which the artefact belongs to (for example, possible versions are 1.0, 2.1, 3.1.2).

The **container-object-id** does not apply to the classes that can be referenced in VTL transformations, therefore is not present in their URN

The **object-id** is the name of the non-maintainable artefact (when the artefact is maintainable its name is already specified as the maintainedobject-id, see above), in particular it has to be specified:

- if the artefact is a `dimension`, `measureDimension`, `timeDimension`, `primaryMeasure` or `dataAttribute` (the object-id is the name of one of the artefacts above, which are data structure components)
- if the artefact is a `concept` (the object-id is the name of the `concept`)

[4] i.e., the artefact belongs to a maintainable class

For example, by using the URN, the VTL transformation that sums two SDMX dataflows DF1 and DF2 and assigns the result to a third persistent dataflow DFR, assuming that DF1, DF2 and DFR are the maintainedobject-id of the three dataflows, that their version is 1.0 and their Agency is AG, would be written as[5]:

```
'urn:sdmx:org.sdmx.infomodel.datastructure.Dataflow=AG:DFR(1.0)' <-
'urn:sdmx:org.sdmx.infomodel.datastructure.Dataflow=AG:DF1(1.0)' +
'urn:sdmx:org.sdmx.infomodel.datastructure.Dataflow=AG:DF2(1.0)'
```

10.2.3 Abbreviation of the URN

The complete formulation of the URN described above is exhaustive but verbose, even for very simple statements. In order to reduce the verbosity through a simplified identifier and make the work of transformation definers easier, proper abbreviations of the URN are allowed. The URN can be abbreviated by omitting the parts that are not essential for the identification of the artefact or that can be deduced from other available information, including the context in which the invocation is made. The possible abbreviations are described below.

- The **SDMXPrefix** can be omitted for all the SDMX objects, because it is a prefixed string (urn:sdmx:org), always the same for SDMX objects.

- The **SDMX-IM-package-name** can be omitted as well because it can be deduced from the class-name that follows it (the table of the SDMX-IM packages and classes that allows this deduction is in the SDMX 2.1 Standards - Section 5 - Registry Specifications, paragraph 6.2.3). In particular, considering the object classes of the artefacts that VTL can reference, the package is:
    - `datastructure` for the classes `dataflow`, `dimension`, `measureDimension`, `timeDimension`, `primaryMeasure`, `dataAttribute`,
    - `conceptscheme` for the classes `concept` and `conceptScheme`
    - `codelist` for the class `codelist`.

- The **class-name** can be omitted as it can be deduced from the VTL invocation. In particular, starting from the VTL class of the invoked artefact (e.g. dataset, component, identifier, measure, attribute, variable, valuedomain), which is known given the syntax of the invoking VTL operator[6], the SDMX class can be deduced from the mapping rules between VTL and SDMX (see the section "Mapping between VTL and SDMX" hereinafter)[7].

---

[5] Since these references to SDMX objects include non-permitted characters as per the VTL ID notation, they need to be included between single quotes, according to the VTL rules for irregular names.

[6] For the syntax of the VTL operators see the VTL Reference Manual

[7] In case the invoked artefact is a VTL component, that can be invoked only within the invocation of a VTL data set (SDMX dataflow), the specific SDMX class-name (e.g. `dimension`, `measureDimension`, `timeDimension`, `primaryMeasure` or `dataAttribute`) can be deduced from the data structure of the SDMX dataflow which the component belongs to.

- If the **agency-id** is not specified, it is assumed by default equal to the agency-id of the `transformationScheme` from which the artefact is invoked. Therefore the agency-id can be omitted if it is the same as the invoking `transformationScheme` and cannot be omitted if the artefact comes from another agency.[8] Take also into account that, according to the VTL consistency rules, the agency of the result of a `transformation` must be the same as its `transformationScheme`, therefore the agency-id can be omitted for all the results (left part of `transformation` statements).

- As for the **maintainedobject-id**, this is essential in some cases while in other cases it can be omitted:
  - if the referenced artefact is a `dataflow`, which is a maintainable class, the maintainedobject-id is the dataflow-id and obviously cannot be omitted;
  - if the referenced artefact is a `dimension`, `measureDimension`, `timeDimension`, `primaryMeasure`, `dataAttribute`, which are not maintainable and belong to the `dataStructure` maintainable class, the maintainedobject-id is the dataStructure-id and can be omitted, given that these components are always invoked within the invocation of a `dataflow`, whose dataStructure-id can be deduced from the SDMX structural definitions;
  - if the referenced artefact is a `concept`, which is not maintainable and belong to the `conceptScheme` maintainable class, the maintained object is the conceptScheme-id and cannot be omitted;
  - if the referenced artefact is a `conceptScheme`, which is a maintainable class, the maintained object is the conceptScheme-id and obviously cannot be omitted;
  - if the referenced artefact is a `codelist`, which is a maintainable class, the maintainedobject-id is the codelist-id and obviously cannot be omitted.

- When the maintainedobject-id is omitted, the **maintainedobject-version** is omitted too. When the maintainedobject-id is not omitted and the maintainedobject-version is omitted, the version 1.0 is assumed by default.

- As said, the **container-object-id** does not apply to the classes that can be referenced in VTL transformations, therefore is not present in their URN

- The **object-id** does not exist for the artefacts belonging to the `dataflow`, `conceptScheme` and `codelist` classes, while it exists and cannot be omitted for the artefacts belonging to the classes `dimension`, `measureDimension`, `timeDimension`, `primaryMeasure`, `dataAttribute` and `concept`, as for them the object-id is the main identifier of the artefact

The simplified object identifier is obtained by omitting all the first part of the URN, including the special characters, till the first part not omitted.

[8] If the Agency is composite (for example AgencyA.Dept1.Unit2), the agency is considered different even if only part of the composite name is different (for example AgencyA.Dept1.Unit3 is a different Agency than the previous one). Moreover the agency-id cannot be omitted in part (i.e., if a `transformationScheme` owned by AgencyA.Dept1.Unit2 references an artefact coming from AgencyA.Dept1.Unit3, the specification of the agency-id becomes mandatory and must be complete, without omitting the possibly equal parts like AgencyA.Dept1)

For example, the full formulation that uses the complete URN shown at the end of the previous paragraph:

‘urn:sdmx:org.sdmx.infomodel.datastructure.Dataflow=AG:DFR(1.0)’ :=
‘urn:sdmx:org.sdmx.infomodel.datastructure.Dataflow=AG:DF1(1.0)’ +
‘urn:sdmx:org.sdmx.infomodel.datastructure.Dataflow=AG:DF2(1.0)’

by omitting all the non-essential parts would become simply:

DFR := DF1 + DF2

The references to the codelists can be simplified similarly. For example, given the non-abbreviated reference to the codelist AG:CL_FREQ(1.0), which is:

‘urn:sdmx:org.sdmx.infomodel.codelist.Codelist=AG:CL_FREQ(1.0)’

if the codelist is referenced from a transformation scheme belonging to the agency AG, omitting all the optional parts, the abbreviated reference would become simply[9]:

CL_FREQ

As for the references to the components, it can be enough to specify the component-Id, given that the dataStructure-Id can be omitted. An example of non-abbreviated reference, if the data structure is DST1 and the component is SECTOR, is the following:

‘urn:sdmx:org.sdmx.infomodel.datastructure.DataStructure=AG:DST1(1.0).SECTOR’

The corresponding fully abbreviated reference, if made from a transformation scheme belonging to AG, would become simply:

SECTOR

For example, the transformation for renaming the component SECTOR of the dataflow DF1 into SEC can be written as[10]:

‘DFR(1.0)’ := ‘DF1(1.0)’ [rename SECTOR to SEC]

The Codes and in general all the Values can be written without any other specification, for example, the transformation to check if the values of the dataflow DF1 are between 0 and 25000 can be written like follows:

‘DFR(1.0)’ := between (‘DF1(1.0)’, 0, 25000)

[9] Single quotes are not needed in this case because CL_FREQ is a VTL regular name.

[10] The result DFR(1.0) is be equal to DF1(1.0) save that the component SECTOR is called SEC

10.3 Mapping between SDMX and VTL

10.3.1 When the mapping occurs

The mapping methods between the VTL and SDMX object classes allow transforming a SDMX definition in a VTL one and vice-versa.

The VTL expressions are accessed through the maintainable artefact "Transformation Scheme" which is composed of "Transformation" nameable artefacts. Each Transformation contains a VTL expression.

Every time a SDMX object is referenced in a VTL Transformation as an input operand, there is the need to generate a VTL definition of the object, so that the VTL operations can take place. This can be made starting from the SDMX definition and applying a SDMX-VTL mapping method in the direction from SDMX to VTL. The possible mapping methods from SDMX to VTL are described in the following paragraphs and are conceived to allow the automatic deduction of the VTL definition of the object from the knowledge of the SDMX definition.

In the opposite direction, every time an object calculated by means of VTL must be treated as a SDMX object (for example for exchanging it through SDMX), there is the need of a SDMX definition of the object, so that the SDMX operations can take place. The SDMX definition is needed for the VTL objects for which a SDMX use is envisaged[11].

The mapping methods from VTL to SDMX are described in the following paragraphs as well, however they do not allow the complete SDMX definition to be automatically deduced from the VTL definition, more than all because the former typically contains additional information in respect to the latter. For example, the definition of a SDMX DSD includes also some mandatory information not available in VTL (like the concept scheme to which the SDMX components refer, the assignmentStatus and attributeRelationship for the DataAttributes and so on). Therefore the mapping methods from VTL to SDMX provide only a general guidance for generating SDMX definitions properly starting from the information available in VTL, independently of how the SDMX definition it is actually generated (manually, automatically or part and part).

10.3.2 General mapping of VTL and SDMX data structures

This section makes reference to the VTL "model for data and their structure" and the correspondent SDMX "Data Structure Definition".

The main type of artefact that the VTL can manipulate is the VTL Data Set, which in general is mapped to the SDMX Dataflow. This means that a VTL Transformation, in the SDMX context, expresses the algorithm for calculating a derived Dataflow starting from some already existing Dataflows (either collected or derived).[12]

---

[11] If a calculated artefact is persistent, it needs a persistent definition, i.e. a SDMX definition in a SDMX environment. Also possible calculated artefact that are not persistent may require a SDMX definition, for example when the result of a non-persistent calculation is disseminated through SDMX tools.

[12] Besides the mapping between one SMDX dataflow and one VTL dataset, it is also possible to map distinct parts of a SDMX dataflow to different VTL datasets, as explained in a following paragraph.

While the VTL transformations are defined in term of Dataflow definitions, they are assumed to be executed on instances of such Dataflows, provided at runtime to the VTL engine (the mechanism for identifying the instances to be processed are not part of the VTL specifications and depend on the implementation of the VTL-based systems). As already said, the SDMX datasets can be considered as instances of SDMX dataflows, therefore a VTL Transformation defined on some SDMX dataflows can be applied on some corresponding SDMX datasets.

A VTL Data Set is structured by one and just one Data Structure and a VTL Data Structure can structure any number of Data Sets. Correspondingly, in the SDMX context a SDMX Dataflow is structured by one and just one DataStructureDefinition and one DataStructureDefinition can structure any number of Dataflows.

A VTL Data Set has a Data Structure made of Components, which in turn can be Identifiers, Measures and Attributes. Similarly, a SDMX DataflowDefinition has a DataStructureDefinition made of components that can be DimensionComponents, PrimaryMeasure and DataAttributes. In turn, a SDMX DimensionComponent can be a Dimension, a TimeDimension or a MeasureDimension. Correspondingly, in the SDMX implementation of the VTL, the VTL Identifiers can be distinguished in three sub-classes (Simple Identifier, Time Identifier or Measure Identifier) even if such a distinction is not evidenced in the VTL IM.

However, a VTL Data Structure can have any number of Identifiers, Measures and Attributes, while a SDMX DataStructureDefinition can have any number of Dimensions and DataAttributes but just one PrimaryMeasure[13]. This is due to a difference between SDMX and VTL in the possible representation methods of the data that contain more measures.

As for SDMX, because the data structure cannot contain more than one measure component (i.e., the `primaryMeasure`), the representation of data having more measures is possible only by means of a particular dimension, called MeasureDimension, which is aimed at containing the name of the measure concept, so that for each observation the value contained in the PrimaryMeasure component is the value of the measure concept reported in the MeasureDimension component.

Instead VTL allows either the method above (an identifier containing the name of the measure together with just one measure component) or a more generic method that consists in defining more measure components in the data structure, one for each measure.

Therefore for multi-measure data more mapping options are possible, as described in more detail in the following sections.

10.3.3 Mapping from SDMX to VTL data structures

10.3.3.1 Basic Mapping

The main mapping method from SDMX to VTL is called **Basic** mapping. This is considered as the default mapping method, applied unless a different method is specified through the `VtlMappingScheme` and `VtlDataflowMapping` classes.

[13] The SDMX community is evaluating the opportunity of allowing more than one measure component in a DataStructureDefinition in the next SDMX version.

When transforming **from SDMX to VTL**, this method consists in leaving the components unchanged and maintaining their names and roles, according to the following table:

SDMX	VTL
Dimension	Simple Identifier
Time Dimension	Time Identifier
Measure Dimension	Measure Identifier
Primary Measure	Measure
Data Attribute	Attribute

According to this method, the resulting VTL structures are always mono-measure (i.e., they have just one measure component) and their Measure is the SDMX PrimaryMeasure. Nevertheless, if the SDMX data structure has a MeasureDimension, which can convey the name of one or more measure concepts, such unique measure component can contain the value of more measures (one for each observation).

As for the SDMX Data Attributes, in VTL they are all considered “at data point / observation level” (i.e. dependent on all the VTL Identifiers), because VTL does not have the SDMX Attribute Relationships, which defines the construct to which the Attribute is related (e.g. observation, dimension or set or group of dimensions, whole data set).

With the Basic mapping, one SDMX observation generates one VTL data point.

10.3.3.2 Pivot Mapping

An alternative mapping method from SDMX to VTL is the **Pivot** mapping, which is different from the Basic method only for the SDMX data structures that contain a MeasureDimension, which are mapped to multi-measure VTL data structures.

The SDMX structures that do not contain a MeasureDimension are mapped like in the Basic mapping (see the previous paragraph).

The SDMX structures that contain a MeasureDimension are mapped as follows (this mapping is equivalent to a pivoting operation):

- A SDMX simple dimension becomes a VTL simple identifier and a SDMX time dimension becomes a VTL time identifier;
- Each possible Concept Cj of the SDMX MeasureDimension is mapped to a VTL Measure, having the same name as the SDMX Concept (i.e. Cj, the VTL Measure is a new component that does not correspond to any component of the SDMX data structure)
- The SDMX MeasureDimension is not mapped to VTL (it disappears in the VTL Data Structure)
- The SDMX PrimaryMeasure is not mapped to VTL as well (it disappears in the VTL Data Structure)
- A SDMX DataAttribute is mapped in different ways according to its AttributeRelationship:
    - If, according to the SDMX attributeRelationship, the values of the DataAttribute do not depend on the values of the MeasureDimension, the SDMX DataAttribute becomes a VTL Attribute having the same name. This happens if the attributeRelationship is not specified (i.e. the Attribute does not depend on any DimensionComponent and

therefore is at data set level), or if it refers to a set (or a group) of dimensions which does not include the MeasureDimension;

- o Otherwise if, according to the SDMX attributeRelationship, the values of the DataAttribute depend on the MeasureDimension, the SDMX Data Attribute is mapped to one VTL Attribute for each possible Concept of the SDMX MeasureDimension; by default, the names of the VTL Attributes are obtained by concatenating the name of the SDMX DataAttribute and the names of the correspondent Concept of the MeasureDimension separated by underscore; for example, if the SDMX DataAttribute is named DA and the possible concepts of the SDMX MeasureDimension are named C1, C2, …, Cn, the corresponding VTL Attributes will be named DA_C1, DA_C2, …, DA_Cn (if different names are desired, they can be achieved afterwards by renaming the Attributes through VTL).

Like in the Basic mapping, the resulting VTL Attributes are considered as dependent on all the VTL identifiers (i.e. “at data point / observation level”), because VTL does not have the SDMX notion of Attribute Relationships.

The summary mapping table from SDMX to VTL for the SDMX data structures that contain a MeasureDimension is the following:

SDMX	VTL
Dimension	Simple Identifier
Time Dimension	Time Identifier
Measure Dimension & Primary Measure	One Measure for each Concept of the SDMX Measure Dimension
Data Attribute not depending on the Measure Dimension	Attribute
Data Attribute depending on the Measure Dimension	One Attribute for each Concept of the SDMX Measure Dimension

Using this mapping method the components of the data structure can change in the conversion from SDMX to VTL and it must be taken into account that the VTL statements can reference only the components of the VTL data structure.

At observation / data point level, calling Cj (j=1, … n) the j^{th} Concept of the MeasureDimension:

- The set of SDMX observations having the same values for all the Dimensions except than the MeasureDimension become one multi-measure VTL Data Point, having one Measure for each Concept Cj of the SDMX MeasureDimension;
- The values of the SDMX simple Dimensions, TimeDimension and DataAttributes not depending on the MeasureDimension (these components by definition have always the same values for all the observations of the set above) become the values of the corresponding VTL simple Identifiers, time Identifier and Attributes.
- The value of the PrimaryMeasure of the SDMX observation belonging to the set above and having MeasureDimension=Cj becomes the value of the VTL Measure Cj
- For the SDMX DataAttributes depending on the MeasureDimension, the value of the DataAttribute DA of the SDMX observation belonging to the set above

and having MeasureDimension=Cj becomes the value of the VTL Attribute DA_Cj

10.3.3.3 From SDMX DataAttributes to VTL Measures

In some cases it may happen that the DataAttributes of the SDMX DataStructure need to be managed as Measures in VTL. Therefore a variant of both the methods above consists in transforming all the SDMX DataAttributes in VTL Measures. When DataAttributes are converted to Measures, the two methods above are called Basic_A2M and Pivot_A2M (the suffix “A2M” stands for Attributes to Measures). As obvious, the resulting VTL data structure is in general multi-measure and does not contain Attributes.

The Basic_A2M and Pivot_A2M behaves respectively like the Basic and Pivot methods, except that the final VTL components which according to the Basic and Pivot methods would have had the role of Attribute assume instead the role of Measure.

Proper VTL features allow changing the role of specific attributes even after the SDMX to VTL mapping: they can be useful when only some of the DataAttributes need to be managed as VTL Measures.

10.3.4 Mapping from VTL to SDMX data structures

10.3.4.1 Basic Mapping

The main mapping method **from VTL to SDMX** is called **Basic** mapping as well.

This is considered as the default mapping method and is applied unless a different method is specified through the VtlMappingScheme and VtlDataflowMapping classes.

The method consists in leaving the components unchanged and maintaining their names and roles in SDMX, according to the following mapping table, which is the same as the basic mapping from SDMX to VTL, only seen in the opposite direction.

This mapping method cannot be applied if the VTL data structure has more than one measure component, given that the SDMX data structure definition allows just one measure component (the PrimaryMeasure). In this case it becomes mandatory to specify a different mapping method through the VtlMappingScheme and VtlDataflowMapping classes.

Mapping table:

VTL	SDMX
Simple Identifier	Dimension
Time Identifier	Time Dimension
Measure Identifier	Measure Dimension
Measure	Primary Measure
Attribute	Data Attribute

If the distinction between simple identifier, time identifier and measure identifier is not maintained in the VTL environment, the classification between Dimension, TimeDimension and MeasureDimension exists only in SDMX, as declared in the DataStructureDefinition.

Regarding the Attributes, because VTL considers all of them “at observation level” as said before, the corresponding SDMX DataAttributes should be put “at observation level” as well (AttributeRelationships referred to the PrimaryMeasure), unless some other information about their AttributeRelationship is available.

Note that the basic mappings in the two directions (from SDMX to VTL and vice-versa) are (almost completely) reversible. In fact, if a SDMX structure is mapped to a VTL structure and then the latter is mapped back to SDMX, the resulting data structure is like the original one (apart for the Attribute relationship, that can be different if the original SDMX structure contains Attributes that are not at observation level). In reverse order, if a VTL structure is mapped to SDMX and then the latter is mapped back to VTL, the original data structure is obtained.

As said, the resulting SDMX definitions must be compliant with the SDMX consistency rules. For example, the SDMX DSD must have the assignmentStatus, which does not exist in VTL, the attributeRelationship for the DataAttributes and so on.

10.3.4.2 Unpivot Mapping

An alternative mapping method from VTL to SDMX is the **Unpivot** mapping.

This mapping method makes sense in case the VTL data structure has more than one measure component (multi-measures VTL structure). For such VTL structures, in fact, the basic method cannot be applied, given that by maintaining the data structure unchanged the resulting SDMX data structure would have more than one measure component, which is not allowed (currently SDMX allows just one measure component, the PrimaryMeasure).

The multi-measures VTL structures have not a Measure Identifier (because the Measures are separate components) and need to be converted to SDMX dataflows having an added MeasureDimension which disambiguates the multiple measures, whose values are all maintained in the primaryMeasure.

The **unpivot** mapping behaves like follows:

- Like in the basic mapping, a VTL simple identifier becomes a SDMX dimension and a VTL time identifier becomes a SDMX time dimension (as said, a measure identifier cannot exist in multi-measure VTL structures);
- a MeasureDimension Component called “measure_name” is added to the SDMX DataStructure;
- a PrimaryMeasure Component called “obs_value” is added to the SDMX DataStructure
- Each VTL Measure is mapped to a Concept of the SDMX MeasureDimension having the same name as the VTL Measure (therefore all the VTL Measure Components disappear in the SDMX DataStructure)
- A VTL Attribute becomes a SDMX DataAttribute having AttributeRelationship referred to all the SDMX Dimensions including the TimeDimension and except the MeasureDimension.

The summary mapping table of the **unpivot** mapping method is the following:

VTL	SDMX
Identifier	Dimension
Time Identifier	Time Dimension

All Measure Components	Measure Dimension (having one Measure Concept for each VTL measure component) & Primary Measure
Attribute	Data Attribute depending on all SDMX dimensions including the TimeDimension and except the MeasureDimension

At observation / data point level:

- a multi-measure VTL Data Point becomes a set of SDMX observations, one for each VTL measure
- the values of the VTL identifiers become the values of the corresponding SDMX Dimensions, for all the observations of the set above
- the name of the j^{th} VTL measure (e.g. “Cj”) becomes the value of the SDMX MeasureDimension of the j^{th} observation of the set (i.e. the concept Cj)
- the value of the j^{th} VTL measure becomes the value of the SDMX PrimaryMeasure of the j^{th} observation of the set
- the values of the VTL Attributes become the values of the corresponding SDMX DataAttributes (in principle for all the observations of the set above)

If desired, this method can be applied also to mono-measure VTL structures, provided that none of the VTL components is mapped to the SDMX measureDimension. Like in the general case, a measureDimension Component called “measure_name” would be added to the SDMX DataStructure and would have just one possible measure concept, corresponding to the unique VTL measure.

In any case, the resulting SDMX definitions must be compliant with the SDMX consistency rules. For example, the possible Concepts of the SDMX MeasureDimension need to be listed in a SDMX ConceptScheme, with proper id, agency and version; moreover the SDMX DSD must have the assignmentStatus, which does not exist in VTL, the attributeRelationship for the DataAttributes and so on.

10.3.4.3 From VTL Measures to SDMX Data Attributes

For the multi-measure VTL structures (having more than one Measure Component), it may happen that the Measures of the VTL DataStructure need to be managed as DataAttributes in SDMX. Therefore a third mapping method consists in transforming one VTL measure in the SDMX primaryMeasure and all the other VTL Measures in SDMX DataAttributes. This method is called M2A (“M2A” stands for “Measures to DataAttributes”).

When applied to mono-measure VTL structures (having one Measure component), the M2A method behaves like the Basic mapping (the VTL Measure component becomes the SDMX primary measure, there is no additional VTL measure to be converted to SDMX DataAttribute). Therefore the mapping table is the same as for the Basic method:

VTL	SDMX
Simple Identifier	Dimension
Time Identifier	Time Dimension
Measure Identifier (if any)	Measure Dimension
Measure	Primary Measure

Attribute	Data Attribute

For multi-measure VTL structures (having more than one Measure component), one VTL Measure becomes the SDMX Primary Measure while the other VTL Measures maintain their names and values but assume the role of DataAttribute in SDMX. The choice of the VTL Measure that correspond to the SDMX primaryMeasure is left to the definer of the SDMX data structure definition.

Taking into account that the multi-measure VTL structures do not have a measure identifier, the mapping table is the following:

VTL	SDMX
Simple Identifier	Dimension
Time Identifier	Time Dimension
One of the Measures	Primary Measure
Other Measures	Data Attribute
Attribute	Data Attribute

Even in this case, the resulting SDMX definitions must be compliant with the SDMX consistency rules. For example, the SDMX DSD must have the assignmentStatus, which does not exist in VTL, the attributeRelationship for the DataAttributes and so on. In particular, the primaryMeasure of the SDMX DSD must be one of the VTL Measures, chosen by the DSD definer.

10.3.5 Declaration of the mapping methods between data structures

In order to define and understand properly VTL transformations, the applied mapping method must be specified. If the default mapping method (Basic) is applied, no specification is needed.

A customized mapping can be defined through the `VtlMappingScheme` and `VtlDataflowMapping` classes (see the section of the SDMX IM relevant to the VTL). A `VtlDataflowMapping` allows specifying the mapping methods to be used for a specific `dataflow`, both in the direction from SDMX to VTL (`toVtlMappingMethod`) and from VTL to SDMX (`fromVtlMappingMethod`).

It is possible to specify the `toVtlMappingMethod` and `fromVtlMappingMethod` also for the conventional `dataflow` called "generic_dataflow": in this case the specified mapping methods are intended to become the default ones, overriding the Basic methods. In turn, the `toVtlMappingMethod` and `fromVtlMappingMethod` declared for a real `artefactName` are intended to override the default ones for such an artefact.

The `VtlMappingScheme` is a container for zero or more `VtlDataflowMapping` `(besides the mappings to artefacts other than the dataflow)`.

A `VtlDataflowMapping` allows associating the URN that identifies a SDMX dataflow to the mapping methods used for it. [14]

[14] The URN can be written either without simplifications or with the simplifications explained in the paragraph "Abbreviations of the URN" below.

10.3.6 Mapping dataflow subsets to distinct VTL data sets

Until now it has been assumed to map one SMDX dataflow to one VTL dataset and vice-versa. This mapping one-to-one is not mandatory according to VTL because a VTL data set is meant to be a set of observations (data points) on a logical plane, having the same logical data structure and the same general meaning, independently of the possible physical representation or storage (see VTL 2.0 User Manual page 24), therefore a SDMX dataflow can be seen either as a unique set of data observations (corresponding to one VTL data set) or as the union of many sets of data observations (each one corresponding to a distinct VTL data set).

As a matter of fact, in some cases it can be useful to define VTL operations involving definite parts of a SDMX dataflow instead than the whole. A typical example of this kind is the validation, and more in general the manipulation, of individual time series belonging to the same dataflow, identifiable through the dimension components of the dataflow except the time dimension. In many cases, these kind of operations can be simplified by mapping, for example, distinct time series (i.e. different parts of a SDMX dataflow) to distinct VTL data sets.

Therefore, in order to make VTL operations simpler when applied on parts of SDMX dataflows, it is allowed to map distinct parts of a SDMX dataflow to distinct VTL data sets according to the following rules and conventions. This kind of mapping is allowed both from SDMX to VTL and from VTL to SDMX, as better explained below.[15]

Hereinafter it has been taken into account that the parts of the SDMX dataflow that map to different VTL datasets must never overlap one another in order to comply with the VTL consistency rules (see also “Transformation Consistency” in the VTL User Manual page 46), i.e. no observation can belong to more than one of these parts.

Given a SDMX dataflow, it is allowed to map to different VTL datasets the groups of observations that have different combination of values for some predefined dimensions, while the observations that have the same combination of values for those dimensions are mapped to the same VTL dataset. For example, assuming that the SDMX dataflow DF1(1.0) has the dimensions INDICATOR, TIME_PERIOD and COUNTRY, and that the user defines the dimensions INDICATOR and COUNTRY as basis for the mapping, all the observations that have the same values for INDICATOR and COUNTRY will be mapped to a specific VTL dataset. This ensures that the different VTL datasets do not overlap one another.

In practice the mapping is obtained like follows:

- For a given SDMX dataflow, the user (VTL definer) defines the dimension components on which the mapping will be based, in a certain order,.[16] Following the example above, imagine that the user declares the dimensions INDICATOR and COUNTRY.

[15] This is an option at disposal of the definer of VTL Transformations; it remains always possible to map one SDMX dataflow to one VTL dataflow and extract the desired parts (e.g. time-series) by means of VTL operators (e.g. “sub”, “filter” …).

[16] This definition is made through the ToVtlSubspace and ToVtlSpaceKey classes and/or in the FromVtlSuperspace and FromVtlSpaceKey classes, depending on the direction of the mapping. When no dimension is declared in such classes, it means that the option of mapping different parts of a SDMX dataflow to different VTL datasets is not used.

- The VTL dataset is given a name composed of the following parts:
    - The reference to a SDMX dataflow (expressed according to the rules described in the previous paragraphs, i.e. URN, abbreviated URN or another alias); for example DF1(1.0);
    - a slash (“/”) as a separator;
    - The reference to a specific part of the SDMX dataflow above, expressed as the concatenation of the values that the predefined SDMX dimensions must have, separated by dots (“.”) and expressed in the order in which the dimensions are defined[17] . For example POPULATION.USA would mean that such a VTL dataset is mapped to the SDMX observations for which INDICATOR is equal to POPULATION and COUNTRY is equal to USA.

In the VTL transformations, this kind of name must be referenced between single quotes because the slash (“/”) is not a regular character according to the VTL rules.

Therefore, the generic name of this kind of VTL datasets would be:

‘DF1(1.0)/*INDICATORValue*.*COUNTRYValue*’

Where *INDICATORValue* and *COUNTRYValue* are placeholders for one value of the INDICATOR and COUNTRY dimensions.

Instead the specific name of one of these VTL datasets would be:

‘DF1(1.0)/POPULATION.USA’

In particular, this is the VTL dataset that contains all the observations of the dataflow DF1 for which MeasureName = POPULATION and COUNTRY = USA.

Let us analyse now what happens in the two directions of the mapping, i.e. from SDMX to VTL and from VTL to SDMX.

As already said, the mapping from SDMX to VTL happens when the VTL dataset is operand of a VTL transformation, instead the mapping from VTL to SDMX happens when the VTL dataset is result of a VTL transformation[18] and need to be treated as a SDMX object. The dimensions on which the mapping is based can be different in the two directions, as defined in the ToVtlSpaceKey class and in the FromVtlSpaceKey class .

First, let us see what happens in the mapping direction from SDMX to VTL, when distinct parts of a SDMX dataflow need to be mapped to distinct VTL datasets that are operand of some VTL transformations.

In order to obtain the VTL data structure from the SDMX one, the SDMX dimensions on which the mapping is based are dropped, then the specified mapping method from SDMX to VTL is applied (i.e. basic, pivot …). The SDMX dimensions on which the mapping is based are not maintained in the VTL data structure because their

---

[17] This is the order in which the dimensions are defined in the ToVtlSpaceKey class or in the FromVtlSpaceKey clase, depending on the direction of the mapping.

[18] It should be remembered that, according to the VTL consistency rules, a given VTL dataset can be the result of no more than one VTL transformation

values are fixed[19]. Naturally, all the VTL datasets obtained from the same SDMX dataflow would have the same VTL data structure.

Taking the example above, for all the datasets of the kind ‘DF1(1.0)/*INDICATORValue*.*COUNTRYValue*’, the dimensions INDICATOR and COUNTRY would be dropped so that the resulting VTL data structure would have only the identifier TIME_PERIOD.

As already said, each VTL dataset is assumed to contain all the observations of the SDMX dataflow having INDICATOR=*INDICATORValue* and COUNTRY= *COUNTRYValue*. For example, the VTL dataset ‘DF1(1.0)/POPULATION.USA’ would contain all the observations of DF1(1.0) having INDICATOR = POPULATION and COUNTRY = USA.

It should be noted that the desired VTL datasets can be obtained also by applying the VTL operator “**sub**” (subspace) to the dataflow DF1(1.0), like in the following VTL expression:

```
‘DF1(1.0)/POPULATION.USA’ :=
DF1(1.0) [ sub  INDICATOR=“POPULATION”, COUNTRY=“USA” ]
```

Therefore, the use of the operator “sub” on a dataflow is a valid alternative to the mapping of different parts of a SDMX dataflow to different VTL datasets in the direction from SDMX to VTL.

Let us now analyse the mapping direction from VTL to SDMX.

In this situation distinct parts of a SDMX dataflow are calculated as distinct VTL datasets, under the constraint that they must have the same VTL data structure.

in order to obtain the SDMX data structure from the VTL one, first the desired mapping method from VTL to SDMX is applied (i.e. basic, unpivot …), then the dimensions on which the mapping is based are added and assigned the corresponding values.

For example, assume that one wants to calculate the dataflow DF2(1.0) with the dimensions INDICATOR, TIME_PERIOD and COUNTRY and that distinct parts of this dataflow, identified through the dimensions INDICATOR and COUNTRY, are calculated through different VTL transformations as distinct VTL datasets, each one having the TIME_PERIOD as the only identifier. The relevant VTL transformations would be of this kind:

‘DF2(1.0)/*INDICATORValue*.*COUNTRYValue*’ := expression

The two values *INDICATORValue* and *COUNTRYValue* would be assigned to the dimensions INDICATOR and COUNTRY respectively, which are in the SDMX data structure but not in the VTL one.

---

[19] Given the VTL consistency rules on the identifiers of the operands, dropping the dimensions having fixed values allows to compose parts of SMDX dataflows coming from different dataflows and having in origin different dimensions, provided that their identifiers become the same in VTL. For example, it becomes possible to compose time series whichever dimensions they originally have, provided that all the dimensions except the date are assigned a fixed value and eliminated.

A specific example of calculation of one of these VTL datasets is the following:

‘DF2(1.0)/GDPPERCAPITA.USA’ := expression

It has been assumed that the expression results in a VTL datasets having the TIME_PERIOD as the only identifier and that, in the mapping from VTL to SMDX, the dimensions INDICATOR and COUNTRY are added to the SDMX data structure and assume the values GDPPERCAPITA and USA respectively.

Assuming that DF1 contains also the GDP in the dimension INDICATOR, the GDPPERCAPITA could be calculated through VTL as follows:

‘DF2(1.0)/GDPPERCAPITA.USA’ :=
‘DF1(1.0)/GDP.USA’ / ‘DF1(1.0)/POPULATION.USA’

‘DF2(1.0)/GDPPERCAPITA.CANADA’ :=
‘DF1(1.0)/GDP.CANADA’ / ‘DF1(1.0)/POPULATION.CANADA’

… … …

All the VTL calculated datasets above will be part of the same calculated SDMX dataflow DF2(1.0).

As an alternative to mapping different parts of a SDMX dataflow to different VTL datasets in the direction from VTL to SDMX, it is possible to use of the VTL operator “**union**”, like in the following example:

DF2(1.0) := union (DF2_1(1.0), … … … , DF2_N(1.0))

In this transformation it has been assumed that the VTL datasets DF2_j(1.0), with j=1…N, have the identifiers TIME_PERIOD, INDICATOR and COUNTRY and have been previously calculated by means of other VTL transformations. If these datasets are calculated without the identifiers INDICATOR and COUNTRY, these can be added by using the VTL operator “**calc**”, for example:

DF2.j(1.0) :=
(‘DF1(1.0)/GDP.USA’ / ‘DF1(1.0)/POPULATION.USA’)
[calc identifier INDICATOR=GDPPERCAPITA, identifier COUNTRY=USA]

When this kind of mapping is used from VTL to SDMX, particular attention has to be given to the consistency of the VTL operations, ensuring that the various parts calculated through different transformations and mapped to the same SDMX dataflow do not overlap and have the same structure.

10.3.7 Mapping variables and value domains between VTL and SDMX

With reference to the VTL “model for Variables and Value domains”, the following additional mappings have to be considered:

VTL	SDMX
Data Set Component	Although this abstraction exists in SDMX, it does not have an explicit

	definition and correspond to a Component (either a Dimension or a PrimaryMeasure or a DataAttribute) belonging to one specific Dataflow[20]
Represented Variable	**Concept** (having a Representation)
Value Domain	**Representation** (see the Structure Pattern in the Base Package)
Enumerated Value Domain / Code List	**Codelist** (for enumerated Dimension, PrimaryMeasure, DataAttribute) or **ConceptScheme** (for MeasureDimension)
Code	**Code** (for enumerated Dimension, PrimaryMeasure, DataAttribute) or **Concept** (for MeasureDimension)
Described Value Domain	**non-enumerated Representation** (having Facets / ExtendedFacets, see the Structure Pattern in the Base Package)
Value	Although this abstraction exists in SDMX, it does not have an explicit definition and correspond to a Code of the Codelist (for enumerated Representations) or to a valid value (for non-enumerated Representations)or to a Concept (for MeasureDimension)
Value Domain Subset / Set	This abstraction does not exist in SDMX
Enumerated Value Domain Subset / Enumerated Set	This abstraction does not exist in SDMX
Described Value Domain Subset / Described Set	This abstraction does not exist in SDMX
Set list	This abstraction does not exist in SDMX

The main difference between VTL and SDMX relies on the fact that the VTL artefacts for defining subsets do not exist in SDMX, therefore the VTL features for referring to predefined subsets are not available in SDMX. These artefacts are the Value Domain Subset (or Set), either enumerated or described, the Set List (list of values belonging to enumerated subsets) and the Data Set Component (aimed at defining the set of values that the Component of a Data Set can take, possibly a subset of the codelist).

[20] Through SDMX Constraints, it is possible to specify the values that a Component of a Dataflow can assume

Another difference consists in the fact that a Value Domain is an identifiable object in VTL either if enumerated or not, while in SDMX the Codelist (corresponding to a VTL enumerated Value Domain) is identifiable, while the SDMX non-enumerated Representation (corresponding to a VTL non-enumerated Value Domain) is not identifiable. As a consequence, the definition of the VTL rulesets, which in VTL can refer either to enumerated or non-enumerated value domains, in SDMX can refer only to enumerated Value Domains (i.e. to SDMX Codelists).

Moreover, it is important to be aware that some VTL operations (for example the binary operations at data set level) are consistent only if the components having the same names in the operated VTL data sets have the same representation (i.e. the same Value Domain as for VTL). For example, it is possible to obtain correct results from the VTL expression

DS_c := DS_a + DS_b (where DS_a, DS_b, DS_c are VTL Data Sets)

if the matching components in DS_a and DS_b (e.g. ref_date, geo_area, sector, obs_value, obs_status in DS_a and in DS_b) refer to the same general representation. In simpler words, DS_a and DS_b must use the same values/codes for the same ref_date, geo_area, sector, obs_value, obs_status in DS_a and in DS_b, otherwise the relevant values would not match and the result of the operation would be wrong.

The property above is not enforced by construction in SDMX, in fact a Component can have different LocalRepresentations in different Data Structure Definitions, even not compatible one another (for example, it may happen that the component geo_area is represented by ISO-alpha-3 codes in DS_a and by ISO alpha-2 codes in DS_b). Therefore, it will be up to the definer of VTL transformations to ensure that the VTL expressions are consistent with the actual representations of the SDMX Components.

10.4 Mapping between SDMX and VTL Data Types

10.4.1 VTL Data types

According to the VTL User Guide the possible operations in VTL depend on the data types of the artefacts. For example, numbers can be multiplied but text strings cannot. In the VTL Transformations, the compliance between the operators and the data types of their operands is statically checked, i.e., violations result in compile-time errors.

The VTL data types are sub-divided in scalar types (like integers, strings, etc.), which are the types of the scalar values, and compound types (like data sets, components, rulesets, etc.), which are the types of the compound structures. See below the diagram of the VTL data types, taken from the VTL User Manual:

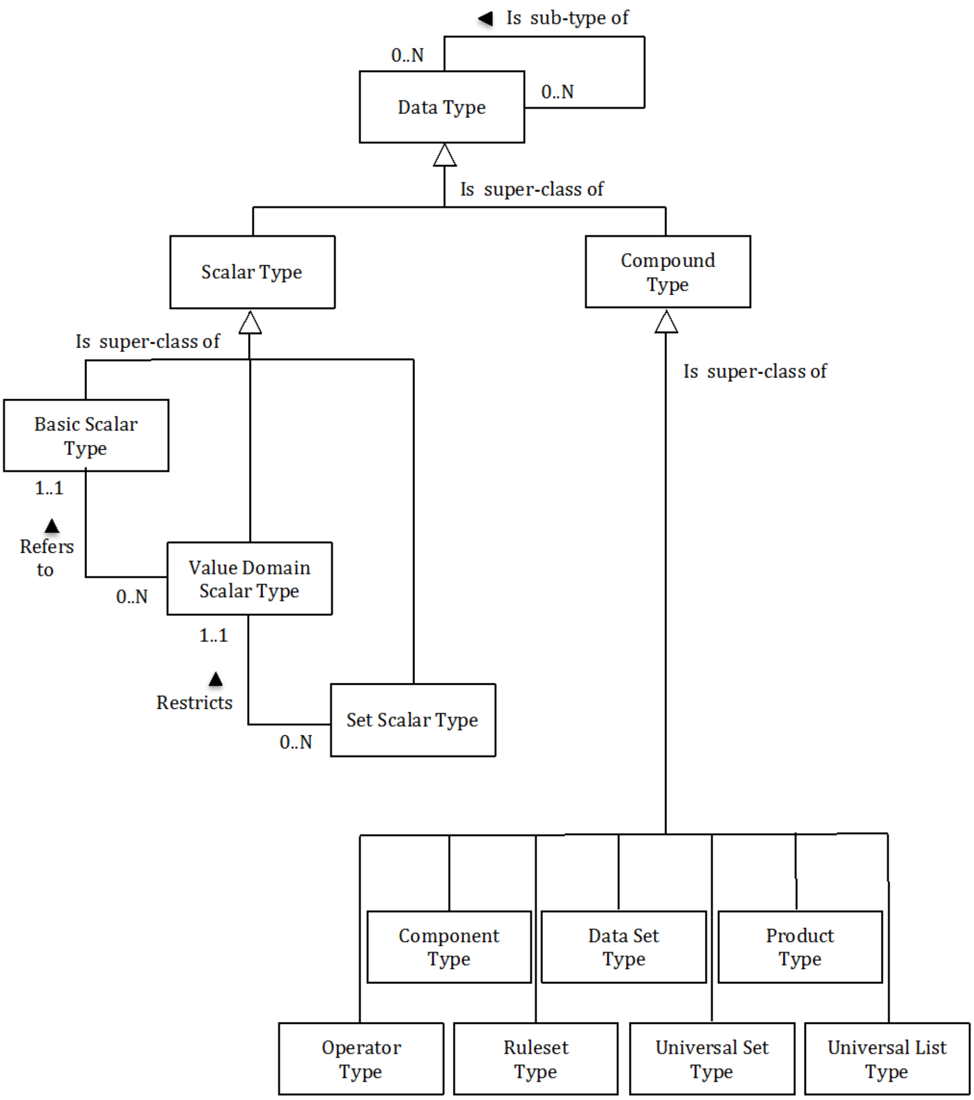

Figure 1 – VTL Data Types

The VTL scalar types are in turn subdivided in basic scalar types, which are elementary (not defined in term of other data types) and Value Domain and Set Scalar types, which are defined in terms of the basic scalar types.

The VTL basic scalar types are listed below and follow a hierarchical structure in terms of supersets/subsets (e.g. “scalar” is the superset of all the basic scalar types:

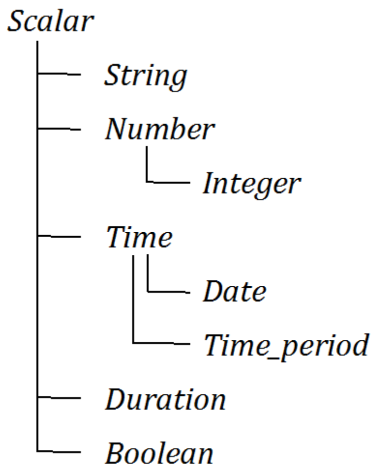

Figure 2 – VTL Basic Scalar Types

10.4.2 VTL basic scalar types and SDMX data types

The VTL assumes that a basic scalar type has a unique internal representation and can have more external representations.

The internal representation is the format used within a VTL system to represent (and process) all the scalar values of a certain type. In principle, this format is hidden and not necessarily known by users. The external representations are instead the external formats of the values of a certain basic scalar type, i.e. the formats known by the users. For example, the internal representation of the dates can be an integer counting the days since a predefined date (e.g. from 01/01/4713 BC up to 31/12/5874897 AD like in Postgres) while two possible external representations are the formats YYYY-MM-GG and MM-GG-YYYY (e.g. respectively 2010-12-31 and 12-31-2010).

The internal representation is the reference format that allows VTL to operate on more values of the same type (for example on more dates) even if such values have different external formats: these values are all converted to the unique internal representation so that they can be composed together (e.g. to find the more recent date, to find the time span between these dates and so on).

The VTL assumes that a unique internal representation exists for each basic scalar type but does not prescribe any particular format for it, leaving the VTL systems free to using they preferred or already existing internal format. By consequence, in VTL the basic scalar types are abstractions not associated to a specific format.

SDMX data types are conceived instead to support the data exchange, therefore they do have a format, which is known by the users and correspond, in VTL terms, to external representations. Therefore, for each VTL basic scalar type there can be more SDMX data types (the latter are explained in the section “General Notes for Implementers” of this document and are actually much more numerous than the former).

The following paragraphs describe the mapping between the SDMX data types and the VTL basic scalar types. This mapping shall be presented in the two directions of possible conversion, i.e. from SDMX to VTL and vice-versa.

The conversion from SDMX to VTL happens when an SDMX artefact acts as inputs of a VTL transformation. As already said, in fact, at compile time the VTL needs to know the VTL type of the operands in order to check their compliance with the VTL operators and at runtime it must convert the values from their external (SDMX) representations to the corresponding internal (VTL) ones.

The opposite conversion, i.e. from VTL to SDMX, happens when a VTL result, i.e. a VTL data set output of a transformation, must become a SDMX artefact (or part of it). The values of the VTL result must be converted into the desired (SDMX) external representations (data types) of the SDMX artefact.

10.4.3 Mapping SDMX data types to VTL basic scalar types

The following table describes the default mapping for converting from the SDMX data types to the VTL basic scalar types.

SDMX data type (BasicComponentDataType)	Default VTL basic scalar type
String (string allowing any character)	**string**
Alpha (string which only allows A-z)	**string**
AlphaNumeric (string which only allows A-z and 0-9)	**string**
Numeric (string which only allows 0-9, but is not numeric so that is can having leading zeros)	**string**
BigInteger (corresponds to XML Schema xs:integer datatype; infinite set of integer values)	**integer**
Integer (corresponds to XML Schema xs:int datatype; between -2147483648 and +2147483647 (inclusive))	**integer**
Long (corresponds to XML Schema xs:long datatype; between -9223372036854775808 and +9223372036854775807 (inclusive))	**integer**
Short (corresponds to XML Schema xs:short datatype; between -32768 and -32767 (inclusive))	**integer**
Decimal (corresponds to XML Schema xs:decimal datatype; subset of real numbers that can be represented as decimals)	**number**
Float (corresponds to XML Schema xs:float datatype; patterned after the IEEE single-precision 32-bit floating point type)	**number**
Double (corresponds to XML Schema xs:double datatype; patterned after the IEEE double-precision 64-bit floating point type)	**number**
Boolean (corresponds to the XML Schema xs:boolean datatype; support the mathematical concept of binary-valued logic: {true, false})	**boolean**
URI (corresponds to the XML Schema xs:anyURI; absolute	**string**

or relative Uniform Resource Identifier Reference)	
Count (an integer following a sequential pattern, increasing by 1 for each occurrence)	**integer**
InclusiveValueRange (decimal number within a closed interval, whose bounds are specified in the SDMX representation by the facets minValue and maxValue)	**number**
ExclusiveValueRange (decimal number within an open interval, whose bounds are specified in the SDMX representation by the facets minValue and maxValue)	**number**
Incremental (decimal number the increased by a specific interval (defined by the interval facet), which is typically enforced outside of the XML validation)	**number**
ObservationalTimePeriod (superset of StandardTimePeriod and TimeRange)	**time**
StandardTimePeriod (superset of BasicTimePeriod and ReportingTimePeriod)	**time**
BasicTimePeriod (superset of GregorianTimePeriod and DateTime)	**date**
GregorianTimePeriod (superset of GregorianYear, GregorianYearMonth, and GregorianDay)	**date**
GregorianYear (YYYY)	**date**
GregorianYearMonth / **GregorianMonth** (YYYY-MM)	**date**
GregorianDay (YYYY-MM-DD)	**date**
ReportingTimePeriod (superset of RepostingYear, ReportingSemester, ReportingTrimester, ReportingQuarter, ReportingMonth, ReportingWeek, ReportingDay)	**time_period**
ReportingYear (YYYY-A1 – 1 year period)	**time_period**
ReportingSemester (YYYY-Ss – 6 month period)	**time_period**
ReportingTrimester (YYYY-Tt – 4 month period)	**time_period**
ReportingQuarter (YYYY-Qq – 3 month period)	**time_period**
ReportingMonth (YYYY-Mmm – 1 month period)	**time_period**
ReportingWeek (YYYY-Www – 7 day period; following ISO 8601 definition of a week in a year)	**time_period**
ReportingDay (YYYY-Dddd – 1 day period)	**time_period**
DateTime (YYYY-MM-DDThh:mm:ss)	**date**
TimeRange (YYYY-MM-DD(Thh:mm:ss)?/<duration>)	**time**
Month (--MM; speicifies a month independent of a year; e.g. February is black history month in the United States)	**string**

MonthDay (--MM-DD; specifies a day within a month independent of a year; e.g. Christmas is December 25th; used to specify reporting year start day)	**string**
Day (---DD; specifies a day independent of a month or year; e.g. the 15th is payday)	**string**
Time (hh:mm:ss; time independent of a date; e.g. coffee break is at 10:00 AM)	**string**
Duration (corresponds to XML Schema xs:duration datatype)	**duration**
XHTML KeyValues IdentifiableReference DataSetReference AttachmentConstraintReference	Metadata type – not applicable Metadata type – not applicable Metadata type – not applicable Metadata type – not applicable Metadata type – not applicable

Figure 14 – Mappings from SDMX data types to VTL Basic Scalar Types

When VTL takes in input SDMX artefacts, it is assumed that a type conversion according to the table above always happens. In case a different VTL basic scalar type is desired, it can be achieved in the VTL program taking in input the default VTL basic scalar type above and applying to it the VTL type conversion features (see the implicit and explicit type conversion and the "cast" operator in the VTL Reference Manual).

10.4.4 Mapping VTL basic scalar types to SDMX data types

The following table describes the default conversion from the SDMX data types to the VTL basic scalar types.

VTL basic scalar type	Default SDMX data type (BasicComponentDataType)	Default output format
string	**String**	Like XML (xs:string)
number	**Float**	Like XML (xs:float)
integer	**Integer**	Like XML (xs:int)
date	**DateTime**	YYYY-MM-DDT00:00:00Z
time	**StandardTimePeriod**	<date>/<date> (as defined above)
time_period	**ReportingTimePeriod (StandardReportingPeriod)**	YYYY-Pppp (according to SDMX)
duration	**Duration**	Like XML (xs:duration) PnYnMnDTnHnMnS
boolean	**Boolean**	Like XML (xs:boolean) with the values "true" or "false"

Figure 14 – Mappings from SDMX data types to VTL Basic Scalar Types

In case a different default conversion is desired, it can be achieved through the CustomTypeScheme and CustomType artefacts (see also the section Transformations and Expressions of the SDMX information model).

The custom output formats can be specified by means of the VTL formatting mask described in the section "Type Conversion and Formatting Mask" of the VTL

Reference Manual. Such a section describes the masks for the VTL basic scalar types “number”, “integer”, “date”, “time”, “time_period” and “duration”. As for the types “string” and “boolean” the VTL conventions are extended with some other special characters as follows.

VTL special characters for the formatting masks	
Number	
D	one numeric digit (if the scientific notation is adopted, D is only for the mantissa)
E	one numeric digit (for the exponent of the scientific notation)
. (dot)	possible separator between the integer and the decimal parts.
, (comma)	possible separator between the integer and the decimal parts.
Time and Duration	
C	century
Y	year
S	semester
Q	quarter
M	month
W	week
D	day
h	hour digit (by default on 24 hours)
m	minute
s	second
d	decimal of second
P	period indicator (see the “duration” codes below)
p	number of periods
AM/PM	indicator of AM / PM (e.g. am/pm for “am” or “pm”)
MONTH	textual representation of the month (e.g., JANUARY for January)
DAY	textual representation of the day (e.g., MONDAY for Monday)
String	
X	any string character
Z	any string character from “A” to “z”
9	any string character from “0” to “9”
Boolean	
B	Boolean using “true” for True and “false” for False
1	Boolean using “1” for True and “0” for False
0	Boolean using “0” for True and “1” for False

The default conversion, either standard or customized, can be used to deduce automatically the representation of the components of the result of a VTL transformation. In alternative, the representation of the resulting SDMX dataflow can be given explicitly by providing its DataStructureDefinition. In other words, the representation specified in the DSD, if available, overrides any default conversion[21].

[21] The representation given in the DSD, if available, must be compatible with the VTL data type, otherwise an error must be raised.

10.4.5 Null Values

The VTL programs can produce in output Null values for Measures and Attributes (Null values are not allowed in the Dimensions).

In the conversions from SDMX to VTL it is assumed by default that a missing value in SDMX becomes a NULL in VTL. Correspondingly, in the conversion from VTL to SDMX it is assumed that a NULL in VTL becomes a missing value in SDMX.

This default assumption can be overridden, separately for each VTL basic scalar type, by specifying which the value that represents the NULL in SDMX is. This can be done through the attribute "nullValue" of the CustomType artefact (see also the section Transformations and Expressions of the SDMX information model).

10.4.6 Format of the literals used in VTL transformations

The VTL programs can contain literals, i.e. specific values of certain data types written directly in the VTL definitions or expressions. The VTL does not prescribe a specific format for the literals and leave the specific VTL systems and the definers of VTL transformations free of using their preferred formats.

Given this discretion, it is essential to know which are the external representations adopted for the literals in a VTL program, in order to interpret them correctly. For example, if the external format for the dates is YYYY-MM-DD the date literal 2010-01-02 has the meaning of 2nd January 2010, instead if the external format for the dates is YYYY-DD-MM the same literal has the meaning of 1st February 2010.

Hereinafter, i.e. in the SDMX implementation of the VTL, it is assumed that the literals are expressed according to the "default output format" of the table of the previous paragraph ("Mapping VTL basic scalar types to SDMX data types") unless otherwise specified.

A different format can be specified in the attribute "vtlLiteralFormat" of the CustomType artefact (see also the section Transformations and Expressions of the SDMX information model).

In case a literal is operand of a VTL Cast operation, the format specified in the Cast overrides all the possible otherwise specified formats.