The following part substitutes the model package n.13 (Transformations and Expressions) of the SDMX 2.1 Information Model (Section 2).

13 Transformations and Expressions

13.1 Introduction

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

The purpose of this model package is to enable the:

- definition of validation and transformation algorithms by means of VTL, in order to specify how to calculate new SDMX 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);
- execution of VTL algorithms, either interpreting the VTL transformations or translating them in whatever other computer language is deemed as appropriate;

This model package does not explain the VTL language or any of the content published in the VTL guides. Rather, this is an illustration of the SDMX classes and attributes that allow defining VTL transformations applied to SDMX artefacts.

The SDMX model represented below is consistent with the VTL 2.0 specification[2]. However, the former uses the SDMX terminology and is a model at technical level (from which the SDMX implementation artefacts for defining VTL transformations are built), whereas the latter uses the VTL terminology and is at conceptual level. The guidelines for mapping these terminologies and using the VTL in the SDMX context can be found in a dedicated chapter ("Validation and Transformation Language") of the Section 6 of the SDMX Standards ("SDMX Technical Notes").

[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.

13.2 Model - Inheritance view

13.2.1 Class Diagram

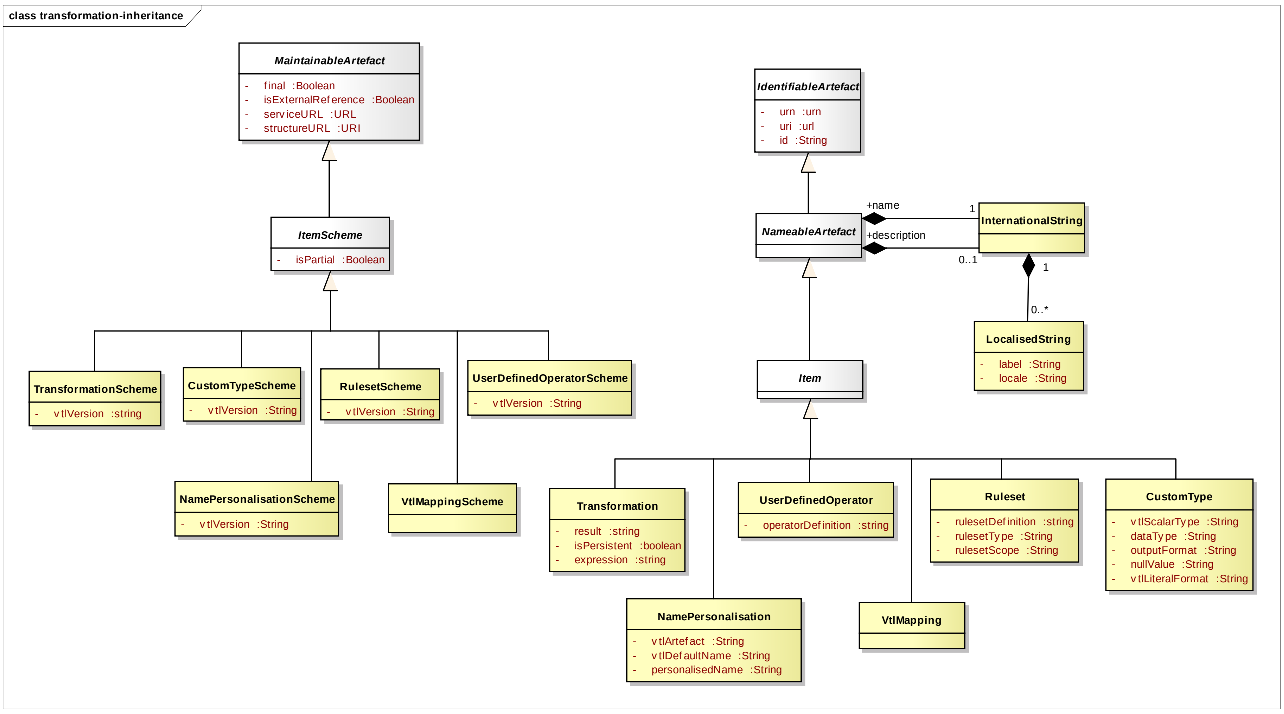

Figure 1: Class inheritance diagram in the Transformations and Expressions Package

13.2.2 Explanation of the Diagram

13.2.2.1 Narrative

The model artefacts `TransformationScheme`, `RulesetScheme`, `UserDefinedOperatorScheme`, `NamePersonalisationScheme`, `CustomTypeScheme`, and `VtlMappingScheme` inherit from `ItemScheme`

These schemes inherit from the *`ItemScheme`* and therefore have the following attributes:

- `id`
- `uri`
- `urn`
- `version`
- `validFrom`
- `validTo`
- `isExternalReference`
- `registryURL`
- `structureURL`
- `repositoryURL`
- `final`
- `isPartial`

The model artefacts `Transformation`, `Ruleset`, `UserDefinedOperator`, `NamePersonalisation`, `VtlMapping`, `CustomType` inherit the attributes and associations of `Item` which itself inherits from `NameableArtefact`. They have the following attributes:

- `id`
- `uri`
- `urn`

The multi-lingual name and description are provided by the relationship to `InternationalString` from `NameableArtefact`.

13.3 Model - Relationship View

13.3.1 Class Diagram

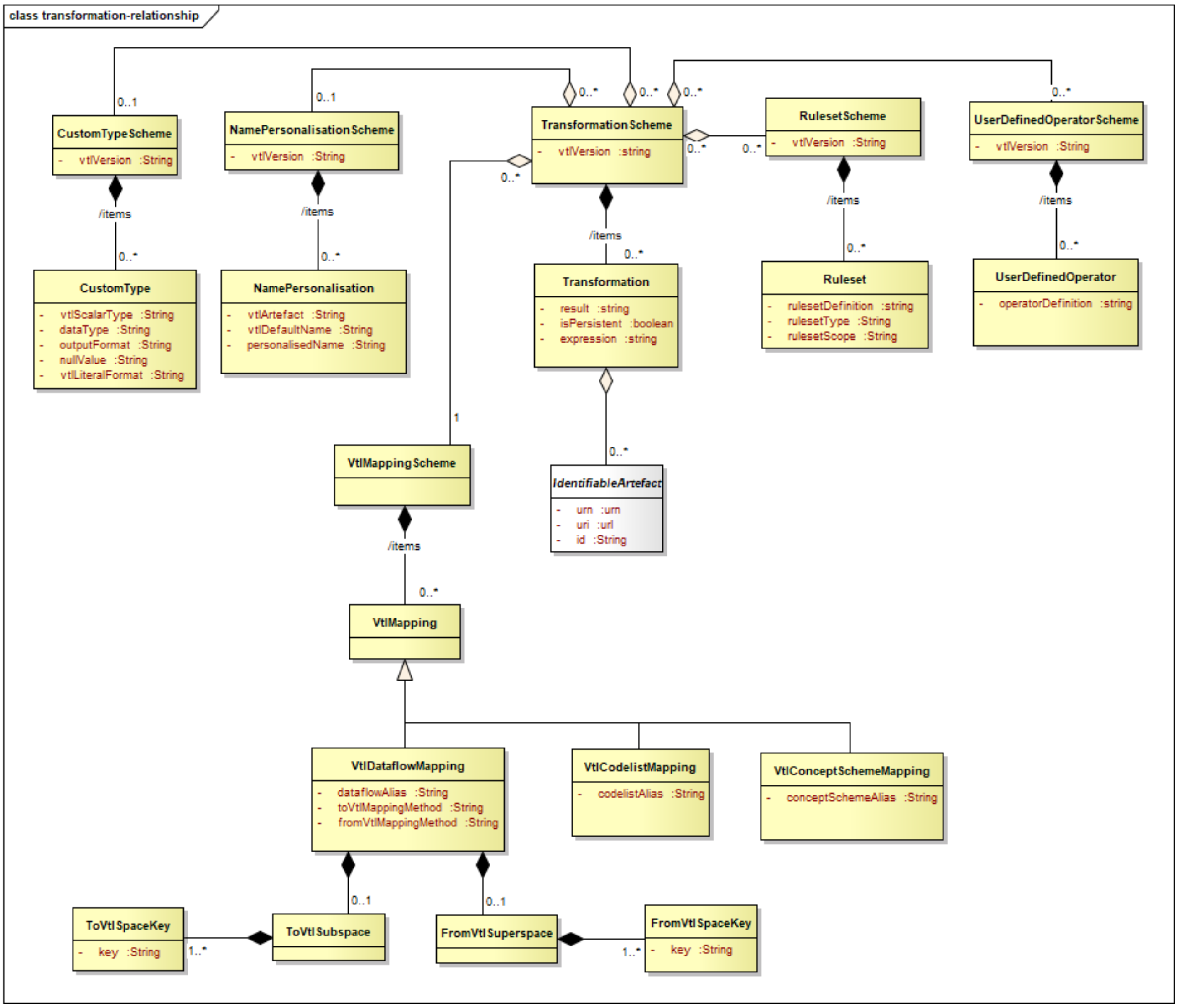

Figure 2: Relationship diagram in the Transformations and Expressions Package

13.3.2 Explanation of the Diagram

13.3.2.1 Narrative - Overview

Transformation Scheme

A `TransformationScheme` is a set of `Transformations` aimed at obtaining some meaningful results for the user (e.g. the validation of one or more Data Sets). This set of `Transformations` is meant to be executed together (in the same run) and may contain any number of transformations in order to produce any number of results. Therefore, a `TransformationScheme` can be considered as a VTL program.

The `TransformationScheme` must include the attribute `vtlVersion` expressed as a string (e.g. "2.0"), as the version of the VTL determines which syntax is used in defining the transformations of the scheme.

A `Transformation` consists of a statement which assigns the outcome of the evaluation of a VTL `expression` to a `result` (an artefact of the VTL Information Model, which in the SDMX context can be a persistent or non-persistent `Dataflow`[3]).

For example, assume that D1, D2 and D3 are SDMX `Dataflows` (called Data Sets in VTL) containing information on some goods, specifically: D3 the current stocks, D1 the stocks of the previous date, D2 the flows in the last period. A possible VTL `Transformation` aimed at checking the consistency between flows and stocks is the following:

Dr := If ((D1 + D2) = D3, then "true", else "false")

In this `Transformation`:

- Dr — is the result (a new dataflow)
- := — is an assignment operator
- If ((D1 + D2) = D3, then "true", else "false") — is the expression
- D1, D2, D3 — are the operands
- If, ( ), +, = — are VTL operators

Therefore, the `Transformation` model artefact contains three attributes:

1. `result`

   The left-hand side of a VTL statement, which specifies the Artefact to which the outcome of the expression is assigned. An artefact cannot be result of more than one `Transformations`.

2. `isPersistent`

   An assignment operator, which specifies also the persistency of the left-hand side. The assignment operators are two, namely **:=** for non-persistent assignment (the result is non-persistent) and **<-** for persistent assignment (the result is persistent).

[3] Or a part of a Dataflow, see the chapter "Validation and Transformation Language" of the Section 6 of the SDMX Standards ("SDMX Technical Notes").

3. expression

The right-hand side of a VTL statement, which is the expression to be evaluated. An expression consists in the invocation of VTL operators in a certain order. When an operator is invoked, for each input parameter, an actual argument is passed to the operator, which returns an actual argument for the output parameter. An expression is simply a text string written according the VTL grammar.

Because an Artefact can be the result of just one Transformation and a Transformation belongs to just one TransformationScheme, it follows also that an Artefact (e.g. a new Dataflow) is produced in just one TransformationScheme.

The result of a Transformation can be input of other Transformations. The VTL assumes that non-persistent results are maintained only within the same TransformationScheme in which they are produced. Therefore, a non-persistent result of a Transformation can be the operand of other Transformations of the same TransformationScheme, whereas a persistent result can be operand of transformations of any TransformationScheme[4].

The TransformationScheme has an association to zero of more RulesetScheme, zero or more UserDefinedOperatorScheme, zero or one NamePersonalisationScheme, zero or one VtlMappingScheme, and zero or one CustomTypeScheme

The RulesetScheme, UserDefinedOperatorScheme NamePersonalisationScheme and CustomTypeScheme have an attribute vtlVersion. Thus, a TransformationScheme using a specific version of VTL can be linked to such schemes only if they are consistent with the same VTL version.

Ruleset Scheme

Some VTL Operators can invoke rulesets, i.e., sets of previously defined rules to be applied by the Operator[5]. Once defined, a Ruleset is persistent and can be invoked as many times as needed. The knowledge of the rulesets' definitions (if any) is essential for understanding the actual behaviour of the Transformations that use them: this is achieved through the RulesetScheme model artefact. The RulesetScheme is the container for one or more Ruleset.

User Defined Operator Scheme

The VTL allows to define UserDefinedOperator (i.e. custom operators) by means of the VTL standard ones. The knowledge of the definitions of the user defined operators (if any) is essential for understanding the actual behaviour of the Transformations that invoke them: this is achieved through the UserDefinedOperatorScheme. The UserDefinedOperatorScheme is a container for zero of more UserDefinedOperator.

Name Personalisation Scheme

In some operations, the VTL assigns by default some standard names to some measures and/or attributes of the data structure of the result[6]. When needed, the VTL allows also to personalise the names to be assigned. The knowledge of the personalised names (if any) is essential for

[4] Provided that the VTL consistency rules are accomplished (see the "Generic Model for Transformations" in the VTL User Manual and its sub-section "Transformation Consistency").

[5] VTL 2.0 has two kind of Rulesets: Datapoint and Hierarchical Rulesets.

[6] For example, the **check** operator produces some new components in the result called by default **bool_var**, **errorcode**, **errorlevel**, **imbalance**. These names can be personalised if needed.

understanding the actual behaviour of the `Transformation`: this is achieved through the `NamePersonalisationScheme`. A `NamePersonalisation` specifies a personalised name that will be assigned in place of a VTL standard name. The `NamePersonalisationScheme` is a container for zero or more `NamePersonalisation`.

VTL Mapping

The mappings between SDMX and VTL can be relevant to the names of the artefacts and to the methods for converting the data structures from SDMX to VTL and vice-versa.

The VTL assumes that the operands are directly referenced in the expressions through their actual names (unique identifiers). In the SDMX implementation of VTL, the SDMX artefacts are referenced through VTL aliases. The `alias` can be the complete URN of the artefact, an abbreviated URN or another user-defined name, as described in the Section 6 of the SDMX Standards ("SDMX Technical Notes"), section "*Mapping between VTL and SDMX*".

The VTL mappings define the correspondence between the aliases and the actual SDMX artefacts, associating an `alias` to each SDMX artefact referenced in the VTL expressions. This correspondence is needed for three kinds of SDMX artefacts: Dataflows, Codelists and Concept Schemes. Therefore, there are three corresponding mapping subclasses: VtlDataflowMapping;, VtlCodelistMapping; VtlConceptSchemeMapping.

As for the Dataflows, it is also possible to specify the mapping method to be applied to convert the Data Structure of the Dataflow. This kind of conversion can happen in two directions, from SDMX to VTL when a SDMX artefact is accessed by a VTL Transformation (`toVtlMappingMethod`), or from VTL to SDMX when a VTL calculated artefact needs a SDMX definition (`fromVtlMappingMethod`).

The default mapping method from SDMX to VTL is called "Basic". Three alternative mapping methods are possible, called "Pivot", "Basic-A2M", "Pivot-A2M" ("A2M" stands for "Attributes to Measures", i.e. the SDMX Data Attributes become VTL Measures).

The default mapping method from VTL to SDMX is also called "Basic", and the two alternative mapping methods are called "Unpivot" and "M2A" ("M2A" stands for "Measures to Attributes", i.e. the first VTL Measure becomes the SDMX primary measure and the other VTL Measures become SDMX Data Attributes).

In both mapping directions, if the default mapping method is used (Basic), no specification is needed. When an alternative mapping method is needed for some artefact, this has to be specified in `toVtlMappingMethod or fromVtlMappingMethod`.

The features above are achieved through the `VtlMappingScheme,` which is a container for zero or more `VtlMapping`.

ToVtlSubspace and ToVtlSpaceKey

Although in general one SDMX Dataflow is mapped to one VTL dataset and vice-versa, it is also allowed to map distinct parts of a single SDMX Dataflow to distinct VTL data sets according to the rules and conventions described in the Section 6 of the SDMX Standards ("SDMX Technical Notes"), section "*Mapping between VTL and SDMX*".

In the direction from SDMX to VTL, this is achieved by fixing the values of some predefined Dimensions of the SDMX Data Structure: all the observations having such combination of values are mapped to one corresponding VTL dataset (the Dimensions having fixed values are not maintained in the Data Structure of the resulting VTL dataset). The `ToVtlSubspace` and

ToVtlSpaceKey classes allow to define these Dimensions. When one SDMX Dataflow is mapped to just one VTL dataset these classes are not used.

FromVtlSuperspace and FromVtlSpaceKey

Analogously, in the direction from VTL to SDMX, it is possible to map more calculated VTL datasets to distinct parts of a single SDMX Dataflow, as long as these VTL datasets have the same Data Structure. This can be done by providing, for each VTL dataset, distinct values for some additional SDMX Dimensions that are not part of the VTL data structure. The FromVtlSuperspace and FromVtlSpaceKey classes allow to define these dimensions. When one VTL dataset is mapped to just one SDMX Dataflow these classes are not used.

Custom Type Scheme

As already said, a Transformation consists of a statement which assigns the outcome of the evaluation of a VTL expression to a result, i.e. an artefact of the VTL Information Model. which in the SDMX context can be a persistent or non-persistent Dataflow[7]. Therefore, the VTL data type of the outcome of the VTL expression has to be converted into the SDMX data type of the result Dataflow. A default conversion table is assumed[8]. The CustomTypeScheme allows to specify custom conversions that override the default conversion table. A CustomType specifies the custom conversion for a VTL scalar type, that will override the default conversion. The CustomTypeScheme is a container for zero or more CustomType. The SDMX data types assume the role of external representations in respect to VTL[9].

Moreover, a VTL expression can contain literals, i.e. specific values of a certain VTL data type written according to a certain format. For example, consider the following Transformation that extracts from the dataflow D1 the observations for which the "reference_date" belongs to the years 2018 and 2019:

Dr := D1 [filter reference_date between 2018-01-01 and 2019-12-31]

In this expression, the two values 2018-01-01 and 2019-12-31 are literals of the VTL "date" scalar type expressed in the format YYYY-MM-DD.

The VTL literals are assumed to be written in the same format specified in the default conversion table mentioned above for the corresponding scalar type. The CustomTypeScheme allows also to specify custom formats for the VTL literals that override the formats specified in the default conversion table.

13.3.2.2 Definitions

Class	Feature	Description
Transformation Scheme	Inherits from *ItemScheme*	Contains the definitions of transformations meant to produce some results and be executed together

[7] Or a part of a Dataflow, see the chapter "Validation and Transformation Language" of the Section 6 of the SDMX Standards ("SDMX Technical Notes").
[8] The default conversion table is described in the chapter "Validation and Transformation Language" of the Section 6 of the SDMX Standards ("SDMX Technical Notes").
[9] About VTL internal and external representations, see the VTL User Manual, section "basic scalar types", page 53.

Class	Feature	Description
	vtlVersion	The version of the VTL language used for defining transformations
Transformation	Inherits from *Item*	A VTL statement which assigns the outcome of an expression to a result.
	result	The left-hand side of the VTL statement, which identifies the result artefact.
	isPersistent	A boolean that indicates whether the result is permanently stored or not depending on the VTL assignment operator.
	expression	The right-hand side of the VTL statement, that is the expression to be evaluated, which includes the references to the operands of the Transformation.
RulesetScheme	Inherits from *ItemScheme*	Container of rulesets.
	vtlVersion	The version of the VTL language used for defining the rulesets
Ruleset	Inherits from *Item*	A persistent set of rules which can be invoked by means of appropriate VTL operators.
	rulesetDefinition	A VTL statement for the definition of a ruleset (according to the syntax of the VTL definition language)
	rulesetType	The VTL type of the ruleset (e.g., in VTL 2.0, datapoint or hierarchical)
	rulesetScope	The model artefact on which the ruleset is defined (e.g., in VTL 2.0, valuedomain or variable)
UserDefinedOperatorScheme	Inherits from *ItemScheme*	Container of user defined operators
	vtlVersion	The version of the VTL language used for defining the user defined operators

Class	**Feature**	**Description**
`UserDefinedOperator`	Inherits from *Item*	Custom VTL operator (not existing in the standard library) that extends the VTL standard library for specific purposes.
	`operatorDefinition`	A VTL statement for the definition of a new operator: it specifies the operator name, its parameters and their data types, the VTL expression that defines its behaviour.
`NamePersonalisationScheme`	Inherits from *ItemScheme*	Container of name personalisations.
	`vtlVersion`	The VTL version which the VTL names to be personalised belong to.
`NamePersonalisation`	Inherits from *Item*	Personalised name that is assigned in place of a VTL standard name in some VTL operations.
	`vtlArtefact`	VTL model artefact to which the VTL standard name to be personalised refers, e.g.variable, value domain.
	`vtlDefaultName`	The VTL standard name to be personalised.
	`personalisedName`	The personalised name to be used in place of the VTL standard name.
`VtlMappingScheme`	Inherits from *ItemScheme*	Container of VTL mappings.

`VtlMapping`	Inherits from
	Item
	
	Sub classes are:
	`VtlDataflowMapping`
	`VtlCodelistMapping`
	`VtlConceptSchemeMapping`
		Single mapping between SDMX and VTL .

`VtlDataflowMapping`	Inherits from *VtlMapping*	Single mapping between SDMX dataflow and VTL dataset
	`dataflowAlias`	Alias used in VTL to reference a SDMX dataflow. The alias must be univocal: different SDMX artefacts cannot have the same VTL alias.

Class	Feature	Description
	`toVtlMappingMethod`	Custom specification of the mapping method from SDMX to VTL data structures.
	`fromVtlMappingMethod`	Custom specification of the mapping method from VTL to SDMX data structures
`ToVtlSubspace`		Subspace of the SDMX dimensions used to identify the parts of the dataflow to be mapped to distinct VTL datasets
`ToVtlSpaceKey`		A SDMX dimension that contributes to identify the parts of the dataflow to be mapped to distinct VTL datasets
	`Key`	The identity of the dimension in the data structure definition that contributes to identify the parts of the dataflow to be mapped to distinct VTL datasets
`FromVtlSuperspace`		Superspace composed of the dimensions added to the SDMX data structure to identify the parts of the dataflow coming from distinct VTL datasets
`FromVtlSpaceKey`		A SDMX dimension that contributes to identify the parts of the dataflow coming from distinct VTL datasets
	`Key`	The identity of the dimension in the data structure definition that contributes to identify the parts of the dataflow coming from distinct VTL datasets
`VtlCodelistMapping`	`Inherits from` *`VtlMapping`*	Single mapping between SDMX codelist and VTL value domain
	`codelistAlias`	Alias used in VTL to reference a SDMX codelist. The alias must be univocal: different SDMX artefacts cannot have the same VTL alias.

Class	Feature	Description
VtlConceptSchemeMapping	Inherits from *VtlMapping*	Single mapping between SDMX concept scheme and VTL value domain
	conceptSchemeAlias	Alias used in VTL to reference a SDMX concept scheme. The alias must be univocal: different SDMX artefacts cannot have the same VTL alias.
CustomTypeScheme	Inherits from *ItemScheme*	Container of custom specifications for VTL scalar types.
	vtlVersion	The VTL version which the VTL scalar types belong to.
CustomType	Inherits from *Item*	Custom specification for a VTL scalar type.
	vtlScalarType	VTL scalar type for which the custom specifications are given.
	outputFormat	Specifies the formatting mask that the VTL scalar type has to assume in order to be converted to the desired external representation, corresponding to the desired SDMX data type (e.g. YYYY-MM-DD, see also the VTL formatting mask in the VTL Reference Manual and the SDMX Technical Notes). It overrides the “Default output format” of the default conversion table (see “Mapping VTL basic scalar types to SDMX data types” in the SDMX Technical Notes).
	dataType	External data type in which the VTL data type has to be converted (e.g. the GregorianDay). It overrides the “Default SDMX data type” of the default conversion table (see “Mapping VTL basic scalar types to SDMX data types” in the SDMX Technical Notes).

Class	Feature	Description
	nullValue	Value to be produced in the output of the conversion when a component of the vtlScalarType has a Null Value. If not specified, no value is produced.
	vtlLiteralFormat	The format in which the literals of the vtlScalarType are expressed in the VTL program (e.g. YYYY-MM-DD, see also the VTL formatting mask in the VTL Reference Manual and the SDMX Technical Notes). It overrides the “Default output format” of the default conversion table (see “Mapping VTL basic scalar types to SDMX data types” in the SDMX Technical Notes).