DPM 2.0 - Refit project
Operations general specification

DOCUMENT VERSION AND STATUS: v.0.7, Public Working Draft

DOCUMENT DATE: 22/05/2023

DPM Operations general specification

Contents

1 Introduction

This is a technical document specifying the Information Model, the Data Types and other technical characteristics of the DPM Operations.

This documentation is mainly addressed to technical audiences with interest in developing engines or in having a better technical understanding of the DPM Operations. The document is not addressed to normal business users that write calculations, although it can be useful for those users with basic knowledge of information models and programming languages.

In any case, this document is complemented by the document describing the operators used for the operations, which should be the main reference for users writing calculations.

1.1 Why

EBA and EIOPA have been using for some years a semi-formal expression language for business users to write and share their data validation requirements.

The fact that is semi-formal allows automating some aspects for translating the expression into another language (e.g., XBRL Formula Link base). But because it is not completely formal, it is, by definition, not possible to create a compiler that fully automates the translation to another language or an interpreter for it.

This document, and its complementary document on the operators, aims to provide this required formalization so that DPM Operations become fully formalised, therefore enabling the automation regarding the calculations written by EBA and EIOPA.

1.2 DPM-XL and DPM-ML

The DPM Operations have two representations: As expressions, with the DPM eXpression Language (DPM-XL) and as a structured representation in the database following a metamodel (DPM Metamodel Language or DPM-ML).

The reason for having **DPM-XL** is that it is the language business users write and can understand. Because it is the input of all the process, it is important to keep this language formal, so that it can be translated and executed. But, at the same time, it must serve for communication, so it has to be business users friendly, implying, among others, that the operands of the language are referred to by using rendering artifacts (tables, row, columns and sheets).

A key requirement for the DPM-XL has been that the language that is currently used should change as little as possible. So, in practice, the formalisation of the language has been a reverse-engineering process, where the starting point where the validations published by the EBA and EIOPA.

The main reasons for having **DPM-ML** are that it is providing a structured version of the calculations, with no need to parse expressions, and that it is referring to variables, which are stable over time, because they represent business concepts, instead of rendering objects, which change often over time and do not hold any business meaning, but just a representation.

The DPM-ML is automatically derived from the DPM-XL and, eventually, can be also derived from languages other than the DPM-XL. DPM-ML relations to languages other than DPM-XL is out of the scope for this documentation. Instead, the documentation provides details on how operations are represented both in DPM-XL and DPM-ML.

The DPM-XL and DPM-ML share the same operators and the same information model.

1.3 How

The formalisation of the language has tree pillars:

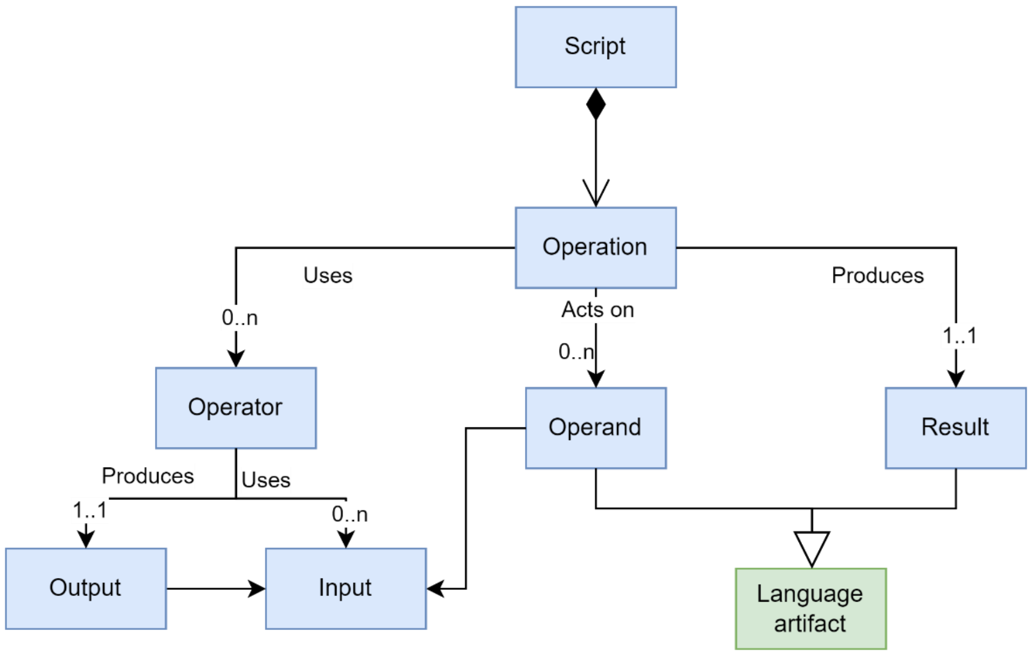
1. **Information model**: Specifies the artefacts that the language is using.
2. **Grammar**: Technical definition of the syntax of the language. Allows developers to build parsers for the language. It is provided in a separate file following the EBNF (Extended Backus-Naur form) notation (Only applicable to DPM-XL).
3. **Semantic specification**: Semantics for all operators of the language, specifying formally their constraints and behaviour.

This document deals with the information model, while the grammar and semantic specification are provided as separate documents.

The Validation and Transformation Language specification has been used as an inspiration for this specification.

2 Information Model

2.1 Operations



The DPM Operations serve to write Scripts, which are computer programs that constitute a run-time and are composed of Operations.

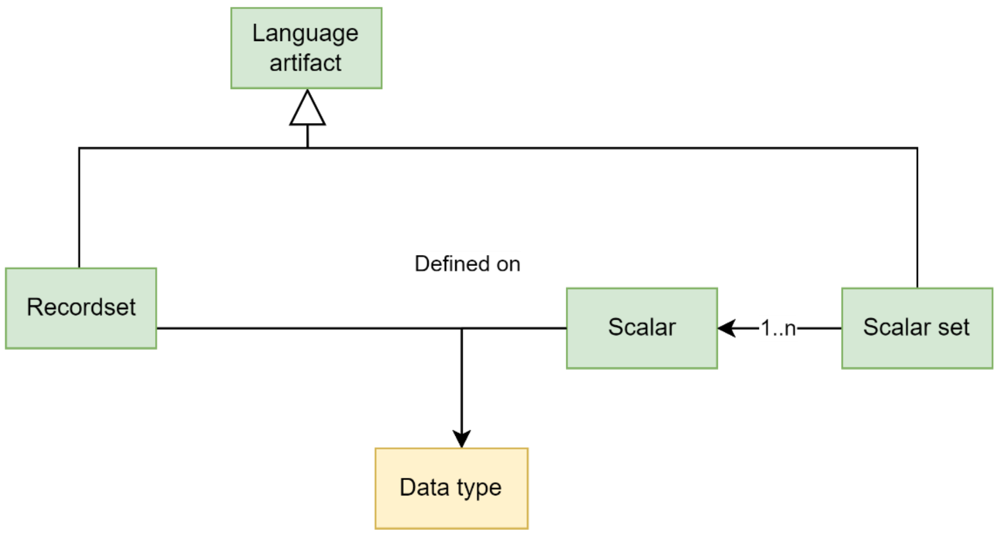
Operations are expressions that use input operands and/or operators to produce a result. Expressions are finite combinations of symbols that are well-formed according to the syntactical rules of the language. Expressions compose some operands in a certain order by means of the operators of the language, to obtain the desired result. The symbols of the expression designate operators, operands, and the order of application of the operators.

Operators specify a type of operation to be performed on some input operands (exceptionally, there may be operators that do not take operands as input, e.g., an operator to get the current time) to generate an output. The output produced by one operator may be used as input for another operator (i.e., operators can be nested).

Operands are specific artifacts from the DPM Expression Language referenced in an expression as input.

The result produced by an operation is also a specific artifact from the DPM Expression Language.

2.2 Language artifacts



DPM Expression Language calculations can operate on and generate as results three different type of Artifacts.

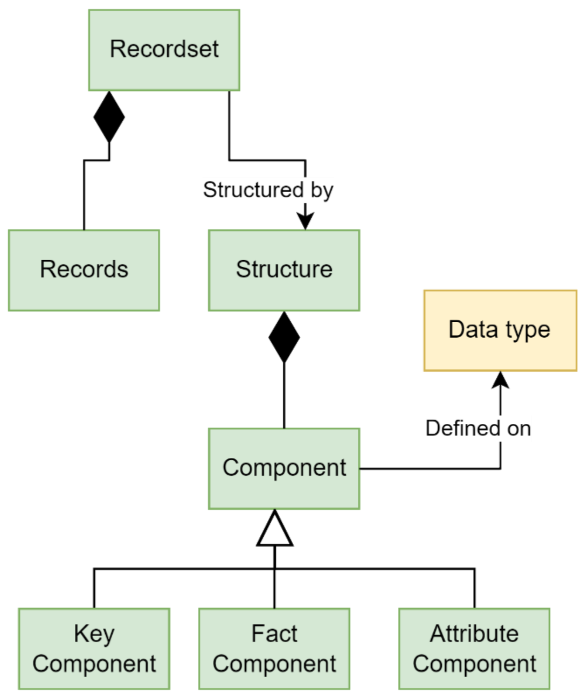
2.2.1 Scalars

Scalars are individual values of a certain *Data Type*.

2.2.2 Scalar sets

Scalar Sets are sets of *Scalar* values defined on the same *Data Type*. *Scalar Sets* are typically used with the *in* operator.

2.2.3 Recordsets



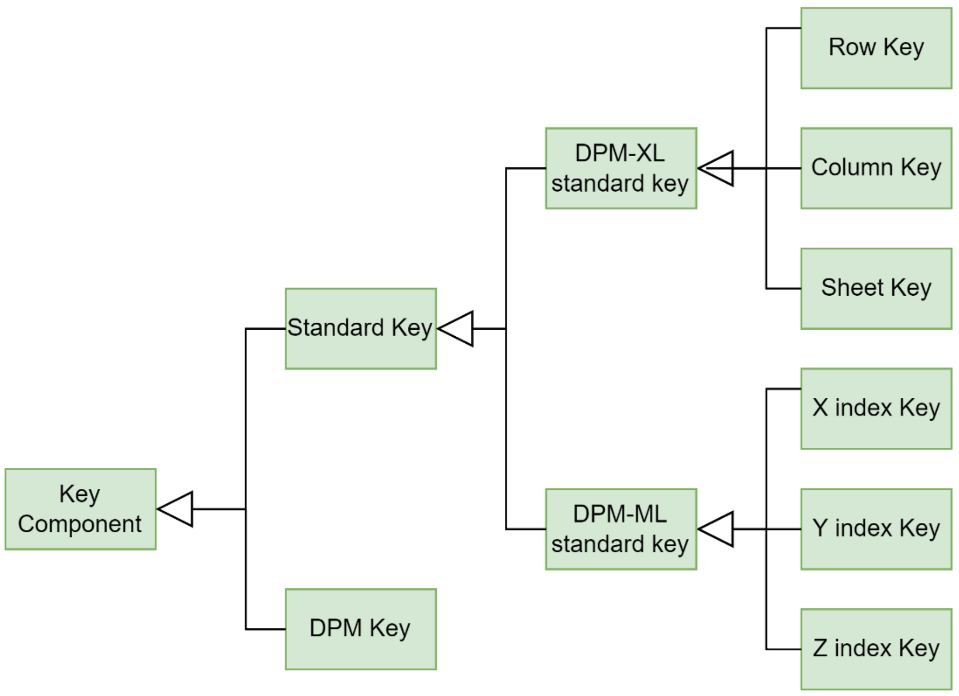
Recordset are collections of *Records* that share a same *Structure*. Technically, *Recordsets* are two-dimensional labelled data structures (tabular), which can be assimilated to Relational Tables or Data Frames. The columns (fields) of the *Recordset* are provided by the *Components* of its *Structure*. The rows of the *Recordset* are its composing *Records.*

The *Structure* of the *Recordset* is a collection of *Components*, which can have one of three roles: *Key, Fact* or *Attribute*. Each *Component* has a name, which must be unique within the *Recordset*. *Key* and *Attribute Components* are defined on a *Data Type*. *Fact Components* may be defined on more than one *Data Type*, when more than one variable is selected. In any case, the *Data Type* is known for each selected *Variable* .

Each Record of the Recordset is individually identified by the combination of the values for its *Key Components*.

A *Recordset* having no *Key Components* behaves like a *Scalar*.

2.2.3.1 *Key components*



Key Component may be *Standard or DPM. Standard Key Components* are different for DPM-ML and DPM-XL, although their behaviour is the same.

Standard Key Components are common to all the *Recordsets*, independently on how the *Variables* are defined in the DPM. For each Recordset, there may be 0 or 1 occurrence of each subtype of *Standard Key* Component.

2.2.3.1.1 Standard DPM-XL Key Components

- *Row Key*: Identifies the *Row Ordinate* from a *Report Table* where the selected *Variable* is located. Arises in *Variable Set Selections*, when more than one *Row* for one *Report Table* is selected. The name for the component is "r". It is defined on the *String Data Type*.
- *Column Key*: Identifies the *Column Ordinate* from a *Report Table* where the selected *Variable* is located. Arises in *Variable Set Selections*, when more than one *Colum* for one *Report Table* is selected. The name for the component is "c". It is defined on the *String Data Type*.
- *Sheet Key*: Identifies the *Sheet Ordinate* from a *Report Table* where the selected *Variable* is located. Arises in *Variable Set Selections*, when more than one *Sheet* for one *Report Table* is selected. The name for the component is "s". It is defined on the *String Data Type*.

2.2.3.1.2 Standard DPM-ML Key Components

Regarding the DPM-ML, the *Standard Components* are three indexes, which do not represent any DPM object. They just serve to make sure that the right variables are matched together when used with an operator. The names for these *Components* are "x", "y" and "z". Note that these indexes do not necessarily correspond to rows, columns, and sheets. During the conversion from DPM-XL, the values of the indexes for each variable shall be provided according to algorithms.

2.2.3.1.3 DPM Key Components

DPM Key Components are specific to how data is defined in the DPM. Arise when *Open Variables* are selected, and a *Recordset* will have one *DPM Key Component* per each *Key Variable* associated to the selected *Variables*.

The name for the *DPM Key Components* is the *Code* of the *Property* associated to the DPM *Key Variable.*

2.2.3.2 Fact Component

A *Recordset* has one mandatory, and only one, *Fact Component*. The name for the component is "f" Its data type depends on the *Data Type* of the selected *Variables*.

If a DPM *Key Variable* is selected, the resulting *Recordset* will have the values for that *Variable* in the *Fact Component*, on top of having them in their corresponding *Key Component*. See example in section 2.2.4.4.

2.2.3.3 Attribute Components

Attribute Components provide additional information to the *Fact* value.

DPM Attribute Components are specific to how data is defined in the DPM. Arise when a *Variable* with associated *Attribute Variable* is selected. A *Recordset* will have one *DPM Attribute Component* per each *Property* or *Metric* associated to any *Attribute Variable* associated to the selected *Variables*.

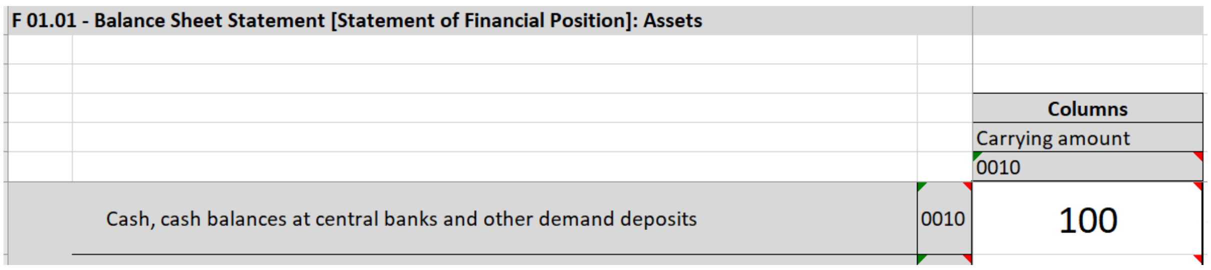
The name for the *DPM Attribute Components* is the *Code* of the *Property* associated to the DPM *Attribute Variable.*

2.2.4 Examples

Recordsets arise from *Open Variable* selections or from selection of multiple *Variables*.

2.2.4.1 Closed Variable Selection

For instance, taking as an example EBA's Table F 01.01:



F 01.01 - Balance Sheet Statement [Statement of Financial Position]: Assets		
		Columns
		Carrying amount
		0010
Cash, cash balances at central banks and other demand deposits	0010	100

{tF_01.01, r0010, c0010}

In the context of a single instance (i.e., disregarding the "ref_date" and the "subject"), selects a *Recordset* without *Key Components*, which works as a scalar with value 100.

In the context of data with multiple dates and/or reference dates, and supposing that the data refer to the subject with code "id123" and to the date 31/12/2020, and that it has been reported with a 1 precision (attribute):

subject	ref_date	f	p
id123	2020-12-31	100	1

2.2.4.2 Open Variable Selection

F 40.01 - Scope of the group: "entity-by-entity"

			Columns		
			Code	Type of code	Entity name
			0011	0015	0031
Rows	Investee	999	123456	LEI	Name1
			123456	ISIN	Name2
			1111	LEI	Name3
			LIN <Key value>	TYC <Key value>	

{F 40.01 c0031}

Selects an *Open Variable*. In this illustrative example, yields a *Recordset* with three *Records*. The *Recordset*, in the context of a single instance, has two *DPM Key Components*, because the selected *Variable* (the one rendered in *Column* 0031 of *Report Table* F 40.01) is associated in the DPM to a *Key* with two *Key Variables*. These *Key Variables* are associated to *Properties* with codes "LIN" (rendered in *Column* 0011*)* and "TYC" (rendered in *Column* 0015).

The following table represents the *Recordset*:

LIN	TYC	f
123456	LEI	Name 1
123456	ISIN	Name 2
1111	LEI	Name 3

2.2.4.3 Variable Set Selection

F 20.05.a - Geographical breakdown of off-balance sheet exposures by residence of the counterparty (a)

ES

			Columns
			Nominal amount
			0010
Rows	Loan commitments given	0010	100
	Financial guarantees given	0020	200
	Other commitments given	0030	300

PT

Columns

			Nominal amount
			0010
Rows	Loan commitments given	0010	400
	Financial guarantees given	0020	500
	Other commitments given	0030	600

{F 20.05 r0020-0030, c0010}

This case selects a set of *Variables* because the selection includes more than one *Row*. This implies that the *Row Key Component* applies to this *Recordset*. Besides, the selected *Variables* are associated to a *Key* with a single *Key Variable*, which is associated to a property with code "RCP".

RCP	r	f
ES	0020	200
ES	0030	300
PT	0020	500
PT	0030	600

In this case, the recordset for the DPM-ML would be slightly different because of the change in the *Standard Key Component*. Instead of the row, we would have any of the three indexes. For instance, using the x index:

RCP	x	f
ES	1	200
ES	2	300
PT	1	500
PT	2	600

Note that the important thing for DPM-ML to be consistent is that all the records with the same row share the same index.

2.2.4.4 Key Variable Selection

F 40.01 - Scope of the group: "entity-by-entity"

			Columns		
			Code	Type of code	Entity name
			0011	0015	0031
Rows	Investee	999	123456	LEI	Name1
			123456	ISIN	Name2
			1111	LEI	Name3

LIN <Key value> TYC <Key value>

{F 40.01 c0015}

Selects an *Open Key Variable*. The *Recordset*, in the context of a single instance, has two *DPM Key Components*, because the selected *Variable* (the one rendered in *Column* 0015 of *Report Table* F 40.01) is associated in the DPM to a *Key* with two *Key Variables* (in fact, it is part of the *Key*). These *Key Variables* are associated to *Properties* with codes "LIN" (rendered in *Column* 0011*)* and "TYC" (rendered in *Column* 0015).

The following table represents the *Recordset*:

LIN	TYC	f
123456	LEI	LEI
123456	ISIN	ISIN
1111	LEI	LEI

Note that the resulting *Recordset* has the values for the type of code two times, one as *Key Component* and another as *Fact Component*. This is necessary to allow doing calculations on the *Fact* while keeping all the *Records* in the *Recordset* uniquely identified.

2.3 Data types

DPM Operations data types are defined with the principle: Create only a data type if they have different behaviour to other data types with at least one operator.

The data types are:

- **String**: Sequence of alphanumeric characters of any length.
- **Number**: Is a rational number of any magnitude and precision, also used as approximation of a real number. Numbers can be treated as points or as intervals. Intervals are defined as a centre plus and minus a radius (c ± r). The centre is the input value provided. The radius is calculated based on the precision of the number as $10^{-p}/2$. .
  - **Integer**: Positive and negative integer numbers and zero. It Is a subtype of the type number. As number subtypes, integers can be treated as intervals.
- **Time interval** denotes time intervals of any duration and expressed with a precision. According to ISO 8601 (ISO standard for the representation of dates and times), a time interval is the intervening time between two time points.
  - **Date** is a subtype of the type time which denotes time points expressed at any precision, which are time intervals starting and ending in the same time point (i.e., intervals of zero duration). A value of type date includes all the parts needed to identify a time point at the desired precision, like the year, the month, the day, the hour, the minute and so on (for example, 2018-04-05 is the fifth of April 2018, at the precision of the day).
  - **Time period** is a subtype of the type time, and denotes non-overlapping time intervals having a regular duration (for example the years, the quarters of years, the months, the weeks and so on). A value of the type time period is composite and must include all the parts needed to identify a regular time period at the desired precision; in particular, the time-period type includes the explicit indication of the kind of regular period considered (e.g., "day", "week", "month", "quarter" …). For example, the value 2018M04, assuming that "M" stands for "month", denotes the month n.4 of the 2018 (April 2018). Moreover, 2018Q2, assuming that "Q" stands

for 1896 “quarter”, denotes the second quarter of 2018. In these examples, the letters M and Q are used to denote the kind of period through its duration.

- **Boolean**: denotes a logical binary state, meaning either “true” or “false”.
- **Category items:** A reference to the code of a Category Item.

Subcategories: A reference to the code of a Subcategory.

2.3.1 Correspondences with DPM data types

The DPM provides a wider list of data types. This is because greater granularity in the data typology may be useful for reporting or representation purposes (e.g., a monetary amount requires a currency attribute, while a pure number does not).

For each DPM data type there is one, and only one DPM Operations data type.

2.3.2 Casting

The casting between data types is possible. Casting can be done explicitly, if the *cast* operator is used, or implicitly when it is allowed. The following table contains all the casting options.

	String	**Number**	**Integer**	**Time interval**	**Date**	**Time period**	**Duration**	**Boolean**	**Item**	**Subcategory**
String		Explicit	Explicit	Explicit	Explicit	Explicit	Explicit	Explicit	Explicit	Explicit
Number	Implicit		Explicit	Not possible	Not possible	Not possible	Not possible	Not possible	Not possible	Not possible
Integer	Implicit	Implicit		Not possible	Not possible	Not possible	Not possible	Not possible	Not possible	Not possible
Time interval	Implicit	Not possible	Not possible		Explicit	Explicit	Not possible	Not possible	Not possible	Not possible
Date	Implicit	Not possible	Not possible	Implicit		Explicit	Not possible	Not possible	Not possible	Not possible
Time period	Implicit	Not possible	Not possible	Implicit	Not possible		Not possible	Not possible	Not possible	Not possible
Duration	Implicit	Not possible	Not possible	Not possible	Not possible	Not possible		Not possible	Not possible	Not possible
Boolean	Implicit	Not possible	Not possible	Not possible	Not possible	Not possible	Not possible		Not possible	Not possible
Item	Implicit	Not possible	Not possible	Not possible	Not possible	Not possible	Not possible	Not possible		Not possible
Subcategory	Implicit	Not possible	Not possible	Not possible	Not possible	Not possible	Not possible	Not possible	Not possible	

2.3.3 Conventions to write scalars

To the effects of DPM-XL, when writing scalars, the following conventions should be followed:

- **String**: Between double quotes (“this is a string”) or single quotes (‘this is also a string’)
- **Number**: Sequence of numeric digits. Dot (“.”) shall be used to separate the decimal and the integer part.
  - **Integer**: Sequence of numeric digits.
- **Time** Follows ISO 8601 (ISO standard for the representation of dates and times). Dates shall be written between hashes (#). A valid time representation is #2021-11-25/2021-12-25#
  - **Date** #2021-11-25#.
  - **Time period** #2021Q4#.
- **Duration** Following ISO8601: P[n]Y[n]M[n]DT[n]H[n]M[n]S. E.g.,: #P3Y6M4DT12H30M5S#
- **Boolean**: *true* or *false*.

- **DPM objects**: A string between brackets with two arguments, first the type and second the code. E.g., [item, eba_RT:x11], [subcategory, eba_CU:iso_currencies]

3 DPM-XL data selection

The data for the expression language is selected by referencing the DPM *Cells* and *Key Variables* of DPM *Report Tables*.

The references to the *Cells* and *Key variables* are done by means of the table codes, ordinate codes and the codes of the *Properties* associated to the *Key Variables*.

The data selection is done by the interaction of the *selection* operator, represented by the curly brackets symbol, and the general scope of an expression, provided by the *with* operator. With this interaction, it is possible to individuate individual DPM *Variables* or sets of them. The actual data are referring to DPM *Variables*, thus making it possible to obtain the data required for the calculations.

The selection of data, together with the DPM definitions, make it possible to determine the *Structure* of the *Recordset* on compile-time.

4 Null in DPM Expression Language

4.1 Null and data types

All the Data Types are assumed to contain the conventional value *null*, which means "no value", or "absence of known value" or "missing value". Note that the *null* value, therefore, is the only value of multiple different types.

For the *String Data Type*, null is considered equivalent to the empty string.

4.2 Implicit an explicit null values

For *Recordsets*, *nulls* may arise in a selection for two reasons:

- **Explicit null**: In the input data for the engine, there is a record without a value.
- **Implicit null:** In the input data for the engine, there is no record for one of the *Variables* in the selection. In this case, the engine will generate a record for that variable with a null value, or another specified default (see *Selection Operator* in part 2).

In any case, there are no differences in how implicit or explicit nulls are treated.

4.3 Null treatment

In general, most of the operations return *null* when any of their arguments is *null*.

- **Comparison operators** (e.g., -, >): if a *null* is involved in the operation, then the result is *null*.
- **Arithmetic operators** (e.g., + , -, *): if a *null* is involved in the operation, then the result is *null*.
- **String operators**: *null* is considered an empty string.
- **Logical operators** (and, or, xor, not): Three-value logic is adopted with the consideration that *null* means unknown. The concrete results for each operator are specified in part 2, in the description of the operators.
- **Conditional operators:** *null* is considered equivalent to *false*.
- **Filtering:** *null* is considered equivalent to *false* (i.e., the *Records* with a *null* value are not selected in the filter).
- **Aggregations** (e.g., sum, avg, max): *nulls* are excluded from the calculations.

- **Intervals:** If the centre is null, the radius is also null.

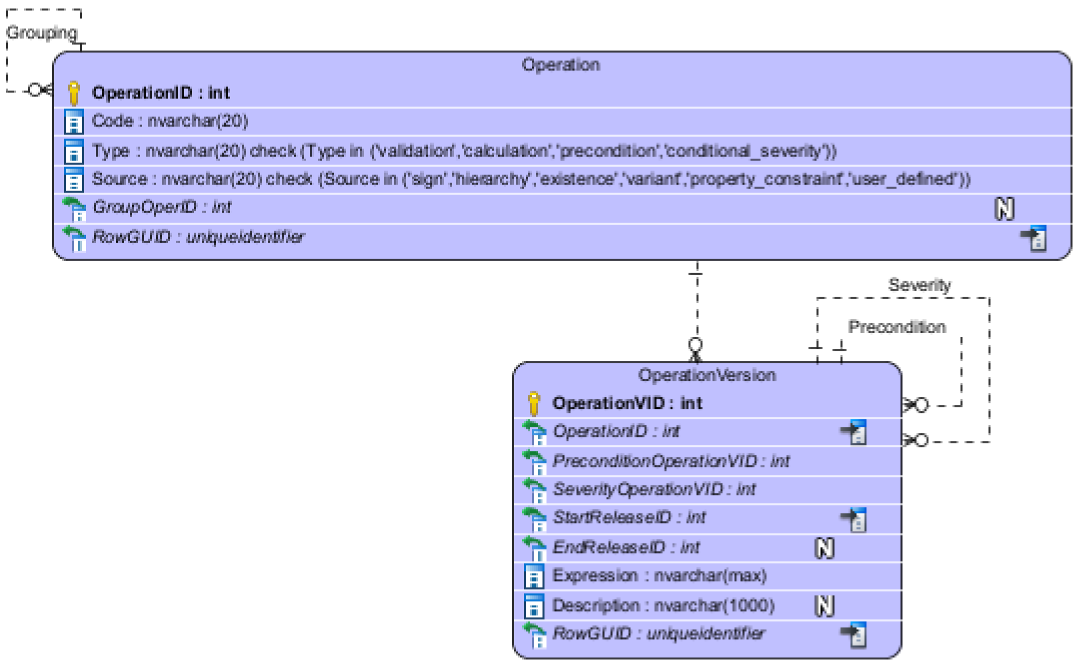
4.4 Overriding the standard null treatment

The standard *null* treatment can be overriding by substituting the *null* values for other values. This can be done in two ways:

- With the *Selection Operator*: It is possible to set a default value in the selection for the *null* values.
- With the nvl *Operator*: serves to substitute nulls for any value.

5 DPM-ML Metamodel

5.1 Operation and OperationVersion


Grouping
Operation
OperationID : int
Code : nvarchar(20)
Type : nvarchar(20) check (Type in ('validation','calculation','precondition','conditional_severity'))
Source : nvarchar(20) check (Source in ('sign','hierarchy','existence','variant','property_constraint','user_defined'))
GroupOperID : int
RowGUID : uniqueidentifier
Severity
Precondition
OperationVersion
OperationVID : int
OperationID : int
PreconditionOperationVID : int
SeverityOperationVID : int
StartReleaseID : int
EndReleaseID : int
Expression : nvarchar(max)
Description : nvarchar(1000)
RowGUID : uniqueidentifier

Operations have a code and can be grouped.

Groups of operations refer to the cases where users define a parent validation and the system derives automatically children validations. There can only be two levels of operations (i.e., a child validation cannot be the parent of another validation). The types of validations that generate children are *Variant* and *Property constraint* validations (see below).

Operations and may be of any of 4 types:

- Validation: Operations that serve to validate the consistency of data. They can provide as output a scalar Boolean value or a recordset with scalar values.
- Calculation: Operations that serve to derive new data. They can provide as output any kind of data.
- Precondition: Operations that serve to determine whether one or more validations shall be applied. They can only provide as output a scalar Boolean value.
- Conditional_severity: Operations that serve to determine the severity of an invalid result for a validation. It can yield a value ‘error’ or ‘warning’

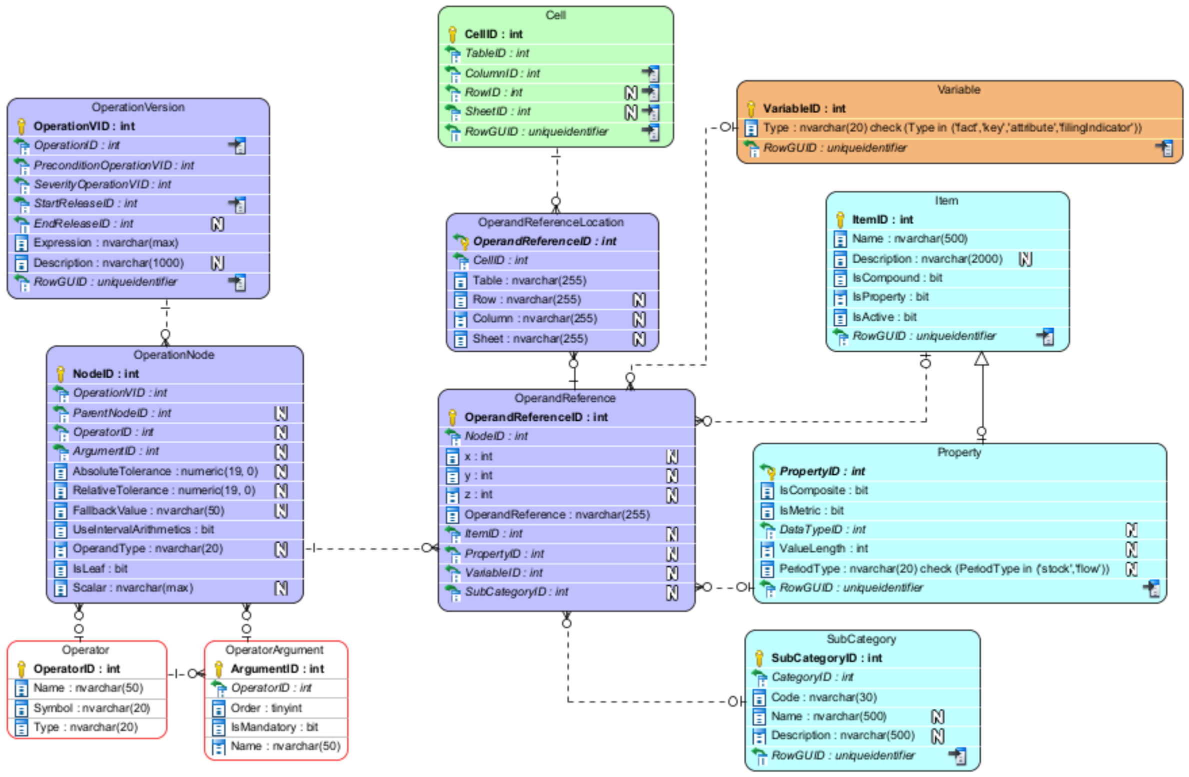
Operations are also classified based on their source for generation as:

- User_defined: Operations that have been defined by users.
- Hierarchy: Validations that have been automatically generated by the system based on the contents of subhierarchies.
- Sign: Validations that have been automatically generated by the system based on the contents of the *Sign* attribute for *TableVersionCells*.
- Existence: Validations that have been automatically generated by the system based on the *IsNullable* attribute for *TableVersionCells*.
- Variant: Validations that are defined at the level of *TableGroups*, including the individual validations generated from them.
- Property_contstraint: Validations that are defined at the level of a property, and from which individual instances of validations based on variables are generated.

Operations can have many versions. A new version of an operation is required whenever any of the following attributes change:

- Expression: The DPM-XL expression of the version.
- Description: Natural language description of what the validation is aimed to check.
- PreconditionOperation: A link to an operation version that acts as precondition.
- SeverityOperation: A link to an operation that determines the conditional severity.

5.2 Operation tree



5.2.1 Operation nodes

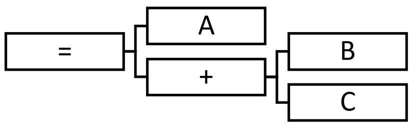
The tree representation of an *OperationVersion* is provided in the *OperationNode* table.

Any *Operation* can be represented as a hierarchical tree, where the arguments of one *Operator* are represented.

For instance, the expression:

$$\mathrm{A} = \mathrm{B} + \mathrm{C}$$

Can be represented as:



Where “=” and “+” are operators, and “A”, “B” and “C” are operands.

This kind of tree is represented in the *OperationNode* table.

5.2.2 Operands’ representation

All *Operands* are represented as *OperationNodes*.

Operands that are *Scalars* not representing a DPM Object (like an *Item*) are completely represented in the *OperationNode* table.

Scalar Operands that are referring to a DPM Object need to be further specified with the tables *OperandSource* and *OperandReference*, which provide the reference to the DPM Object.

Operands that are *Variables* and *Sets of Variables* need also to be further specified with the tables *OperandSource* and *OperandReference*. Concretely, for *Sets of Variables*, it is important to determine the index of each *Variable* in the context of the *Operand*, to be able to match the variables to which the operator applies.

5.2.3 Representation example

For instance, suppose a table (Tab1) with three rows (100, 200 and 300), two columns (100 and 200) and two sheets.

Sheet	A

	Column 100	**Column 200**
Row 100	1	4
Row 200	2	5
Row 300	3	6

Sheet	B

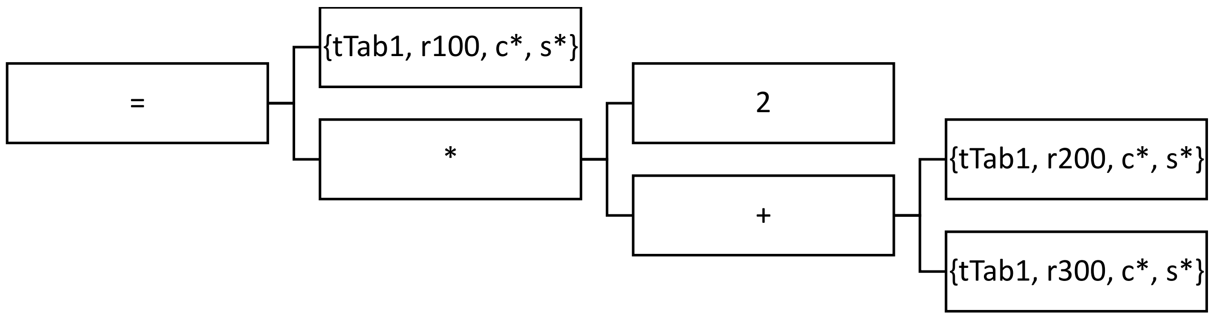
	Column 100	**Column 200**
Row 100	7	10
Row 200	8	11
Row 300	9	12

Note that each number in the cells represents the ID of the corresponding *Variable*.

The following expression:

$$\{\mathrm{tTab1, r100, c^*, s^*}\} = 2 * (\{\mathrm{tTab1, r200, c^*, s^*}\} + \{\mathrm{tTab1, r300, c^*, s^*}\})$$

Would be represented as the following tree:



The representation in the DB would be as follows (only the relevant fields are shown, and for operators and argument simbols are used):

OperationNode table

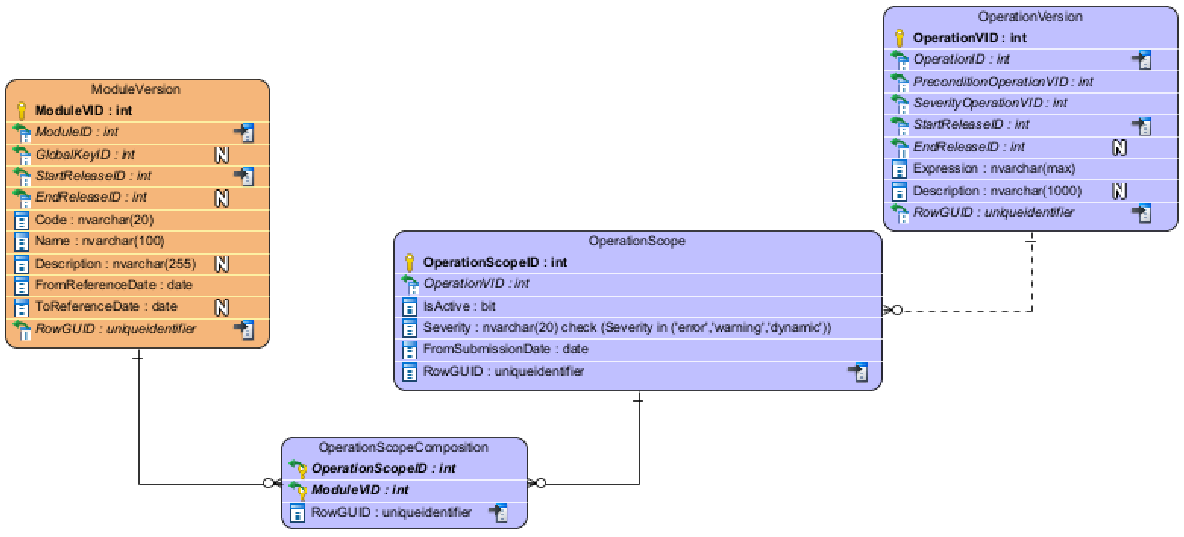
NodeId	ParentNodeID	Operator	Argument	OperandSourceId	Scalar
1		=			
2	1		left	oprnd1	
3	1	*	right		
4	3		left		2
5	3	+	right		
6	5		left	oprnd2	
7	5		right	oprnd3	

OperandReference table

NodeID	x	y	z	VariableID
2		1	1	1
2		2	1	4
2		1	2	7
2		2	2	10
6		1	1	2
6		2	1	5
6		1	2	8
6		2	2	11
7		1	1	3
7		2	1	6
7		1	2	9
7		2	2	12

Note that the operations shall be applied to the variables having the same index. For instance, Variable 1 = 2 * (Variable 2 + Variable 3), because all of them have the index y=1 and z=1.

5.3 Operation Scope

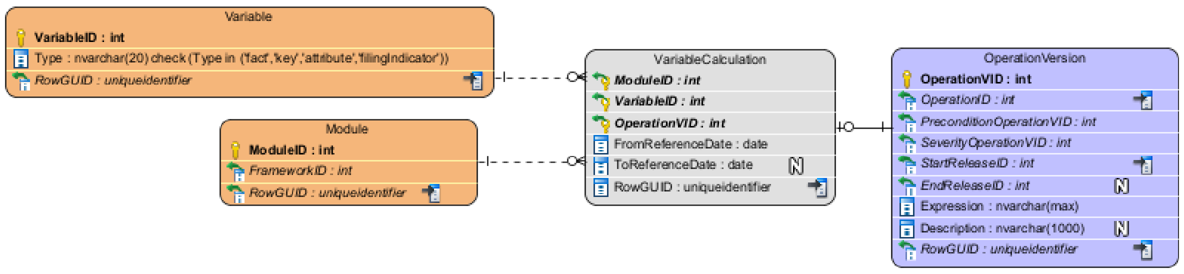


Each operation version can be applied in different scopes. An operation scope refers to the individual module versions or sets of module versions (for cross-module validations) to which the operation logically applies.

For each operation scope, there may be different values for the attributes:

- IsActive: Determines whether the operation shall be run for a certain scope.
- Severity: Determines the severity of the error, if the validation is not passed.
- FromSubmissionDate: Sets a date from which the validation applies to subsequent submissions.

5.4 Variable calculation


Variable
VariableID : int
Type : nvarchar(20) check (Type in ('fact','key','attribute','filingIndicator'))
RowGUID : uniqueidentifier
Module
ModuleID : int
FrameworkID : int
RowGUID : uniqueidentifier
VariableCalculation
ModuleID : int
VariableID : int
OperationVID : int
FromReferenceDate : date
ToReferenceDate : date
RowGUID : uniqueidentifier
OperationVersion
OperationVID : int
OperationID : int
PreconditionOperationVID : int
SeverityOperationVID : int
StartReleaseID : int
EndReleaseID : int
Expression : nvarchar(max)
Description : nvarchar(1000)
RowGUID : uniqueidentifier

For calculations (i.e., operations that serve to calculate values for variables), it is necessary to link the operation to the actual variable that is generated. This is done with the *VariableCalculation* table, which links to the *Variable* and the *Module*. The link to the *Module* is necessary due to the fact that a *Variable* can be used in several *Modules*, being calculated in some cases and not calculated in other cases.