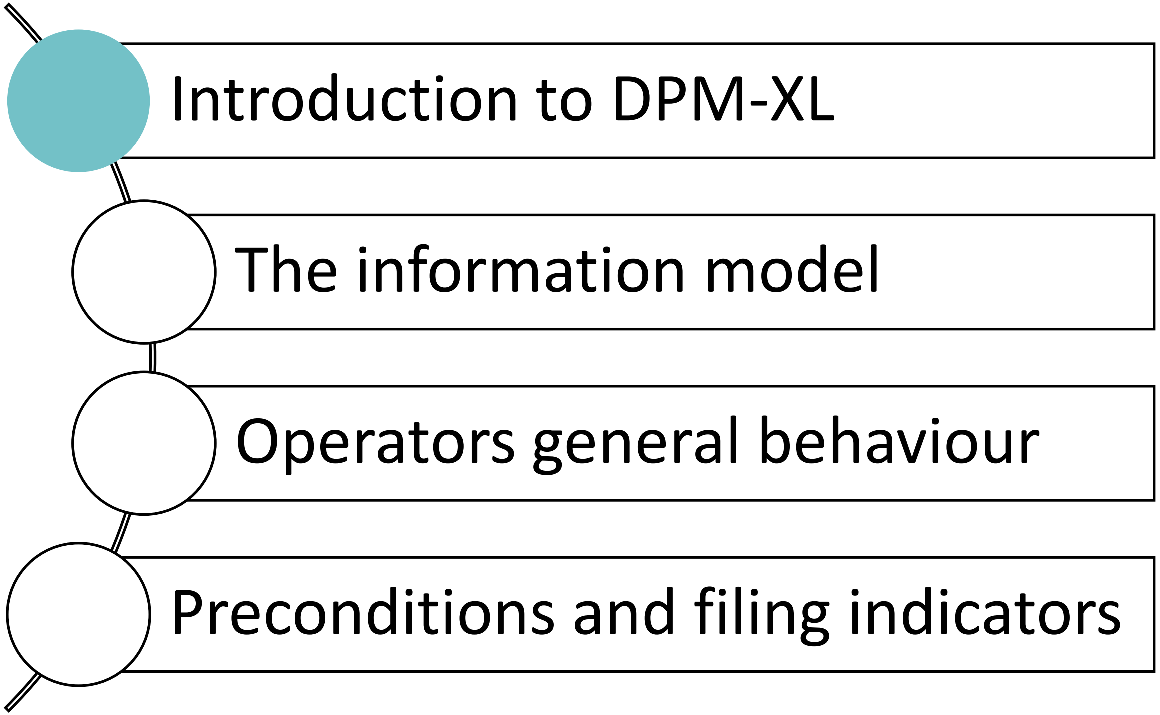
eba European Banking Authority

DPM-XL Introduction

Outline



Introduction to DPM-XL

The information model

Operators general behaviour

Preconditions and filing indicators

What is DPM-XL

DPM-XL is a **language** to write **validation** rules and other **transformations** referring to DPM objects.

It is **based** on the syntax that EBA and EIOPA have been using for years.

It is **formal**, which implies:

- It is executable
- It is testable
- It can be translated automatically to other languages (e.g., XBRL)

About DPM-ML

DPM-ML is a **translation** of DPM-XL into a **database** structure

Operands are **variables**, instead of references to tables and headers

It is automatically **generated** from DPM-XL

There is no need for **business users** to understand DPM-ML

An example of a validation: DPM-XL

C 90.00 - Trading book and market risk thresholds (TBT)

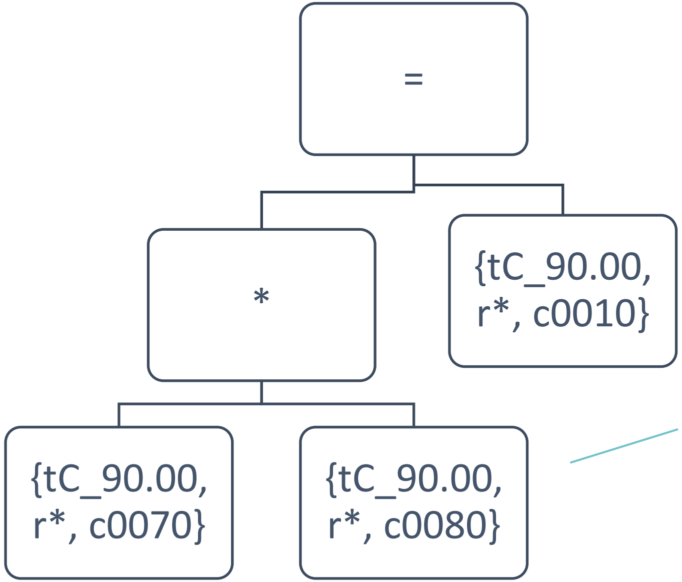
			Columns		
			On - and off - balance sheet business subject to market risk		Total assets
				In % of total assets	
			0010	0070	0080
Rows	Month 3	0010	51	5%	1020
	Month 2	0020	42	4%	1010
	Month 1	0030	60	6%	1000

```
with {tC_90.00, r*}: {c0070} * {c0080} = {c0010}
```

Tree representation of DPM-XL expressions

Any DPM-XL expression can be represented as a tree:

```
with {tC_90.00, r*}: {c0070} * {c0080} = {c0010}
```



=

*

{tC_90.00, r*, c0010}

{tC_90.00, r*, c0070}

{tC_90.00, r*, c0080}

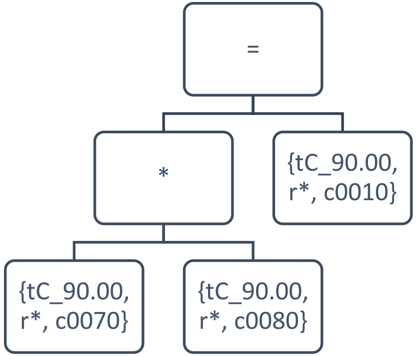
Represents three Variables

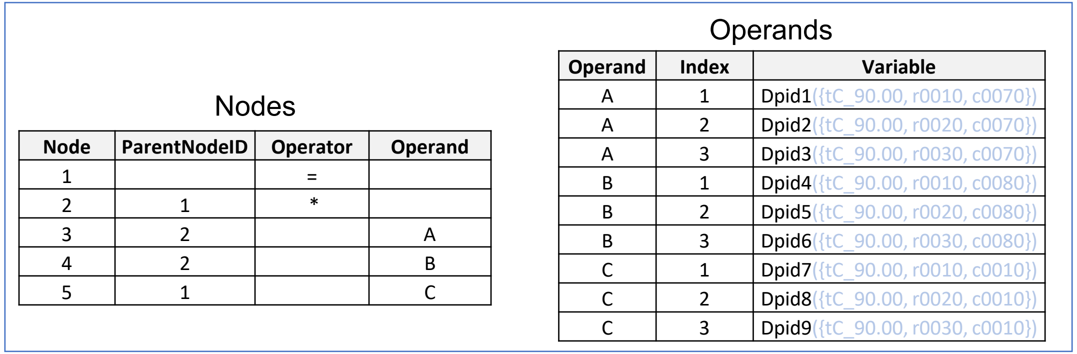
Represents three Variables

Represents three Variables

An example of validation: DPM-ML

The tree of the expression can be then represented in the DB, with reference to actual DPM variables.





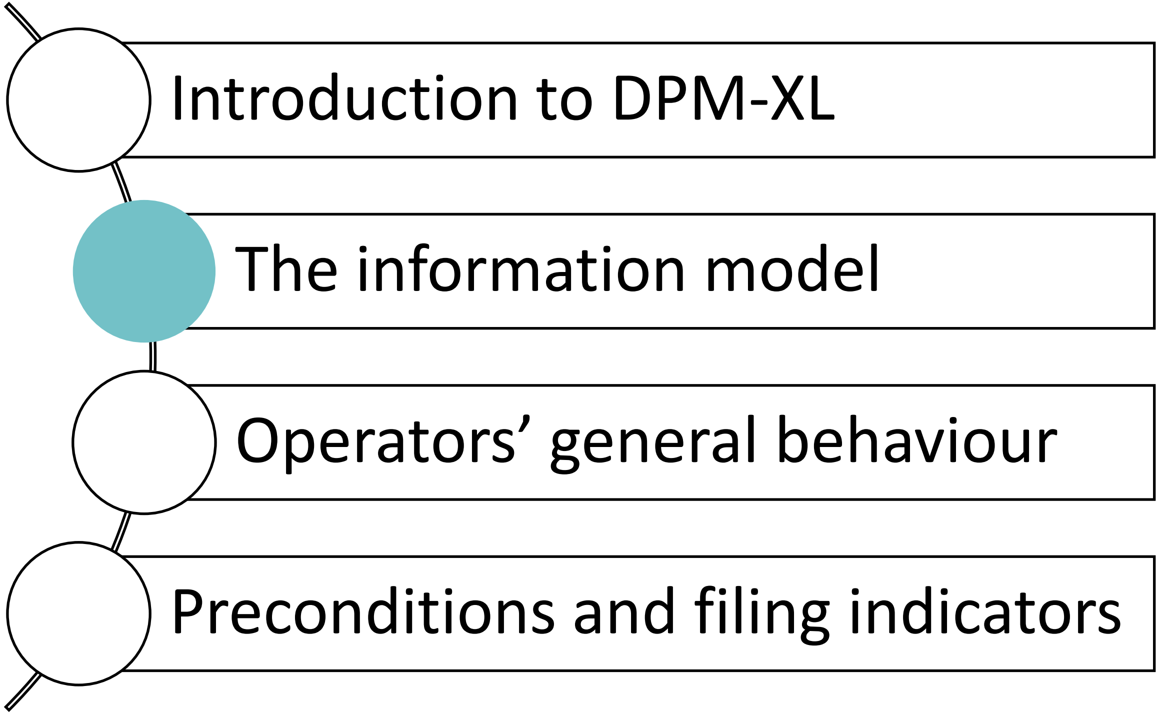
Nodes

Node	ParentNodeID	Operator	Operand
1		=	
2	1	*	
3	2		A
4	2		B
5	1		C

Operands

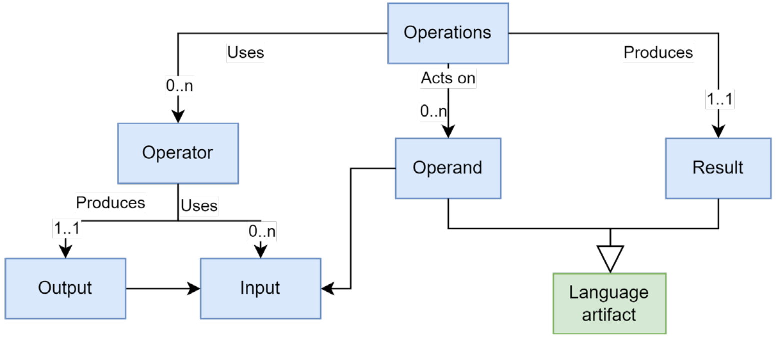
Operand	Index	Variable
A	1	Dpid1({tC_90.00, r0010, c0070})
A	2	Dpid2({tC_90.00, r0020, c0070})
A	3	Dpid3({tC_90.00, r0030, c0070})
B	1	Dpid4({tC_90.00, r0010, c0080})
B	2	Dpid5({tC_90.00, r0020, c0080})
B	3	Dpid6({tC_90.00, r0030, c0080})
C	1	Dpid7({tC_90.00, r0010, c0010})
C	2	Dpid8({tC_90.00, r0020, c0010})
C	3	Dpid9({tC_90.00, r0030, c0010})

Outline



- Introduction to DPM-XL
- The information model
- Operators’ general behaviour
- Preconditions and filing indicators

Information model - Operations



The DPM Expression Language serves to write **operations**.

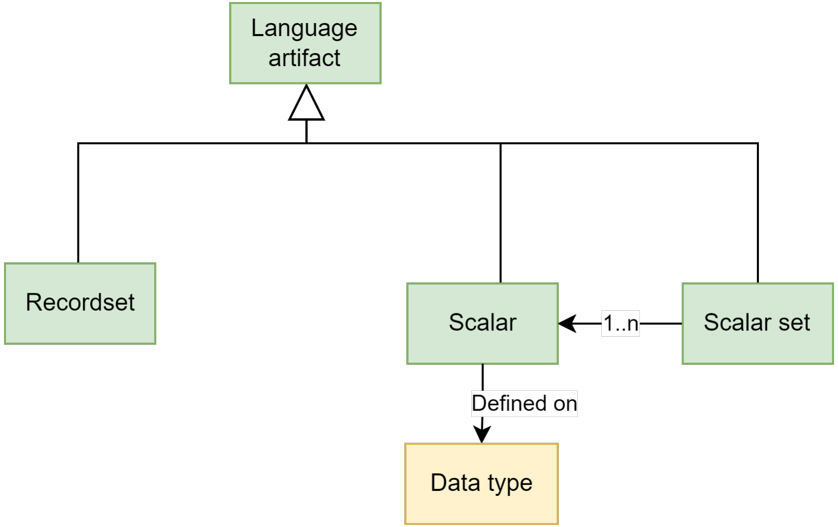
Operations are expressions that use input operands and/or operators to produce a result. Expressions are finite combinations of symbols that are well-formed according to the syntactical rules of the language. Expressions compose some **operands** in a certain order by means of the **operators** of the 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 a calculation is also a specific artifact from the DPM Expression Language.

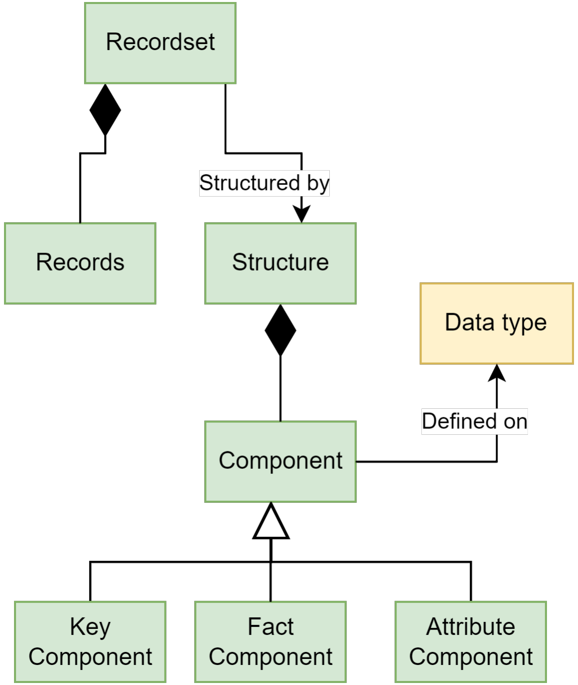
Information model – Language artifacts



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

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

Information model – Recordsets



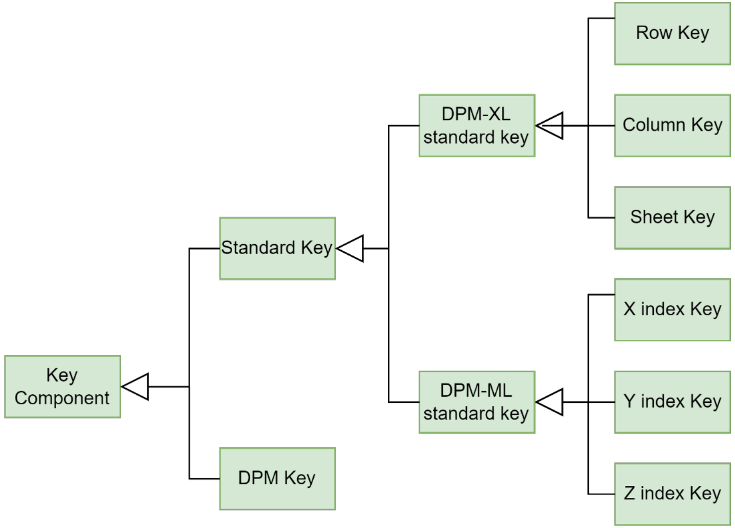
Recordset are collections of *Records* that share a same *Structure*. Technically, *Recordsets* are two-dimensional labelled data structures (tabular), which can be assimilated to Relational Tables or Data Frames. The columns (fields) of the *Recordset* are provided by the ***Components*** of its *Structure*.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

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*.

Information model – Components



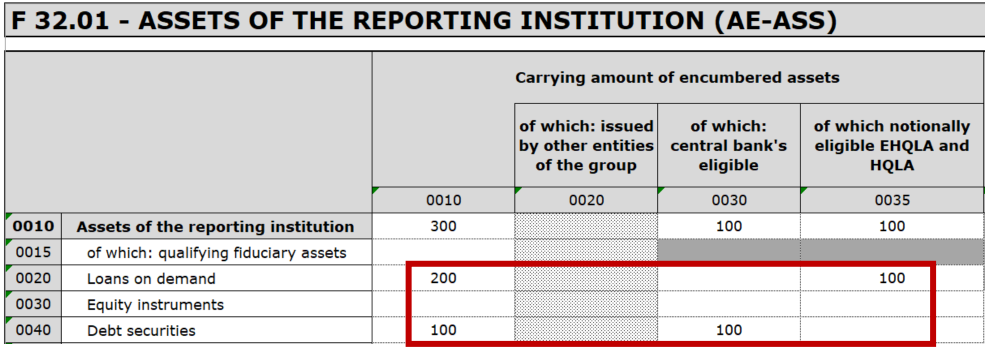
Standard Key Components are common to all the *Recordsets*, independently on how the *Variables* are defined in the DPM. For each Recordset, there may be 0 or 1 occurrence of each subtype of *Standard Key* Component.

- *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*.

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.*

Information model – Recordset Example I



F 32.01 - ASSETS OF THE REPORTING INSTITUTION (AE-ASS)

		Carrying amount of encumbered assets			
			of which: issued by other entities of the group	of which: central bank's eligible	of which notionally eligible EHQLA and HQLA
		0010	0020	0030	0035
0010	**Assets of the reporting institution**	300		100	100
0015	of which: qualifying fiduciary assets				
0020	Loans on demand	200			100
0030	Equity instruments				
0040	Debt securities	100		100	

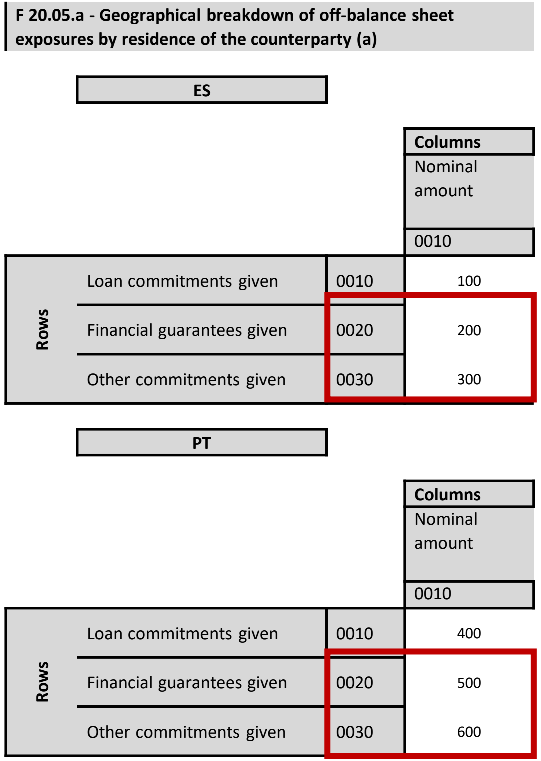
```
{tF_32.01, r0020-0040, (c0010, c0030, c0035)}
```

r	c	f
0020	0010	200
0030	0010	
0040	0010	100
0020	0030	
0030	0030	
0040	0030	100
0020	0035	100
0030	0035	
0040	0035	

```
{tF_32.01, r0020, c0010}
```

f
200

Information model – Recordset Example II



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

ES

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

PT

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

```
{tF_20.05, r0020-0030, c0010}
```

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

Information model – Recordset Example III

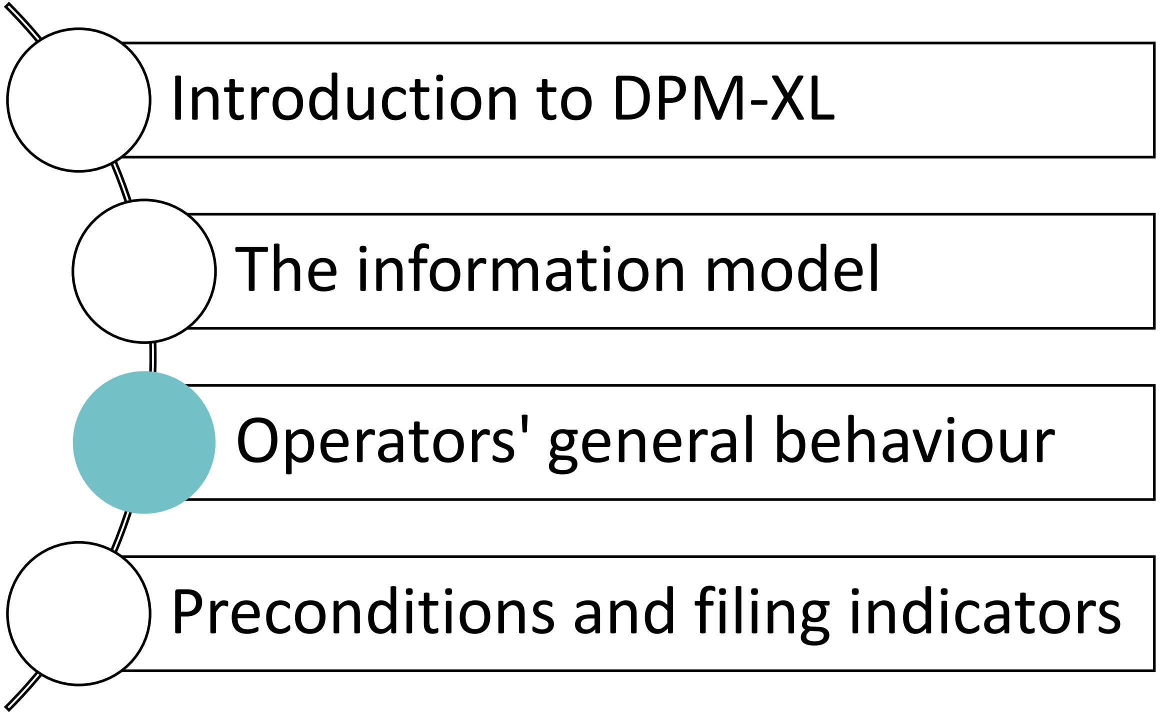
F 40.01 - Scope of the group: “entity-by-entity”

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

```
{tF_40.01, c0031}
```

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

Outline



Introduction to DPM-XL

The information model

Operators' general behaviour

Preconditions and filing indicators

Operators

Selection

- Selection operator
- With

Arithmetic

- Unary plus (+)
- Addition (+)
- Unary minus (-)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Absolute value (abs)
- Exponential (exp)
- Natural logarithm (ln)
- Power (power)
- Logarithm (log)
- Square root (sqrt)

Comparison

- Equal to (=)
- Not equal to (<>)
- Greater than (>, >=)
- Less than (<, <=)
- Element of (in)
- Math characters (match)
- Is null (isnull)

Logical operators

- Conjunction (and)
- Disjunction (or)
- Exclusive disjunction (xor)
- Negation (not)

Aggregate operators

- Sum (sum)
- Count (count)
- Minimum value (min)
- Maximum value (max)
- Average (avg)
- Median value (median)

Conditional operators

- If then else (if)
- Null substitute (nvl)
- Filter (filter)

String operators

- Length (len)
- Concatenate (&)

Time operators

- Time shift (time_shift)

Clause operators

- Where (where)
- Rename (rename)
- Get (get)

The selection operator

- The symbol for the selection operator is the Curly brackets ({})

- The selection operator has three **parts**:

Recordset selection. By referencing:

- Cells (table, rows, columns and/or sheets).
- Variables: References to variable codes.
- Operations: References to the results of other operations.
- Table groups: (Used by EIOPA).

Default value

- Sets a default value in case the selection has missing data or explicit nulls for a data instance.

Interval

- For numeric variables, selects whether the data should be considered as interval or point.

The selection operator: Null values

- The *Recordset* resulting from a selection is composed by all defined ***Variables*** in the selection.
- If one *Variable* is **not reported** (missing), then the *Record* will be equivalent to existing with *null* value.
- For **open tables**, only explicitly reported combinations of key dimensions are considered.

The selection clause – Example

F 01.01 - Balance Sheet Statement [Statement of Financial Position]: Assets

			Columns
			Carrying amount
			0010
Rows	Cash, cash balances at central banks and other demand deposits	0010	**3000**
	Cash on hand	0020	**1000**
	Cash balances at central banks	0030	**2000**
	Other demand deposits	0040	

```
{tF_01.01, r0010-0040,
c0010}
```

r	f
0010	3000
0020	1000
0030	2000
0040	

```
{tF_01.01, r0010-0040,
c0010, default:0}
```

r	f
0010	3000
0020	1000
0030	2000
0040	0

```
{tF_01.01, r0010-0040,
c0010, default:0,
interval:true}
```

r	f
0010	3000±500
0020	1000±500
0030	2000±500
0040	0

Selection and null with open keys

F 40.01 - Scope of the group: “entity-by-entity”

		Columns			
		Code	Type of code	Entity name	Entry date
		0011	0015	0031	0040
Rows	Investee	123456	LEI	Name1	2010-02-01
		123456	ISIN	Name2	
		1111	LEI	Name3	2007-04-03

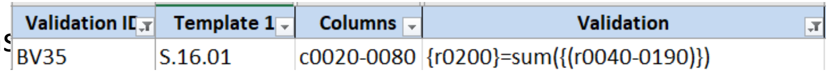
LIN <Key value>   TYC <Key value>

```
{tF_40.01, c0040}
```

LIN	TYC	f
123456	LEI	2010-02-01
123456	ISIN	
1111	LEI	2007-04-03

The with clause

- The with clause serves to provide a common context to all the selections of an expression.
- In the current Excel files, represented in separate columns



Validation ID	Template 1	Columns	Validation
BV35	S.16.01	c0020-0080	{r0200}=sum({(r0040-0190)})

- The with clause uses the following syntax:

```
with partial_selection: expression
```

  - **partial_selection**: Is the selection that is completing the selections in the expression , using the selection clause.
  - **expression**: It is an expression containing selection operators.

- The with clause does not produce an output but modifies the selections in the expression according to some rules. The operator does not produce a node in DPM-ML.
- The selection in the with applies to all selections in the expression unless they are overridden.

The with clause – Examples

```
with {tF_01.01, c0010, default:0, interval:false}:
   {r0010} = {r0020} + {r0030} + {r0040}
```

No operand in the expression overrides the with context.

```
with {tF_01.01, c0010, default:0, interval:false}:
    {r0010} + {r0040} = {tF_04.01, r0010, c0010}
```

The third operand in the expression overrides the table and the column of the with context

```
with {tF_01.01, c0010, default:0, interval:false}:
   {tF_01.01, r0010} + {tF_01.01, r0040} = {tF_04.01, r0010, c0010, default:null}
```

All three operands in the expression override the table in the context. The third operand overrides also the column and the default.

```
with {c0010, default:0, interval:false}:
    {tF_01.01, r0010} + {tF_01.01, r0040} = {tF_04.01, r0010}
```

No operand in the expression overrides the with context

General behaviour for binary operators – Example 1 (recordset and scalar)

```
{tS.26.01, r0600, (c0060, c0080)}
```

c	f
0060	100
0080	200

```
0.25 * {tS.26.01, r0600, (c0060, c0080)}
```

c	f
0060	25
0080	50

General behaviour for binary operators – Example 2 (two recordsets)

```
{tF_04.02.01, r0120, c0010-0020}
```

c	f
0010	100
0020	200

```
{tF_04.02.01, r0140, c0010-0020}
```

c	f
0010	300
0020	400

```
with {tF_04.02.01, c0010-0020)}: {r0120} + {r0140}
```

c	f
0010	400
0020	600

```
with {tF_04.02.01)}: {r0120, c0010-0020} + {r0140, c0030-0040}
```

```
with {tF_04.02.01)}: {r0100-0120, c0010} + {r0140, c0030-0040}
```

General behaviour for binary operators – Example 3 (two recordsets open key)

```
{tC_28.00, c040}
```

INC	f
123	1000
456	2000
789	3000

```
{tC_28.00, c0190}
```

INC	f
123	-100
456	-200
789	-300

```
{tC_28.00, c040} + {tC_28.00, c190}
```

INC	f
123	900
456	1800
789	2700

General behaviour for binary operators – Example 4 (two recordsets, subset identifiers)

```
{tF_40.01, c0110}
```

LIN	TYC	f
123	x1	1
456	x1	0.8
789	x1	0.4

```
{tF_40.02, c0060}
```

LIN	TYC	STC	LHC	LHO	f
123	x1	111	ABC	x1	0.3
123	x1	111	DEF	x1	0.7
456	x1	222	ABC	x1	0.85

```
{tF_40.01, c0110} >= {tF_40.02, c0060}
```

LIN	TYC	STC	LHC	LHO	f
123	x1	111	ABC	x1	true
123	x1	111	DEF	x1	true
456	x1	222	ABC	x1	false

General behaviour for binary operators

- If the two *Operands* of a binary *Operator* are ***Scalars***, the result shall be the *Scalar* resulting of applying the *Operator* to the *Operands*.

- A binary *Operator* applied to a ***Recordset*** *Operand* and a ***Scalar***, will result in a *Recordset* with the same *Structure* as the input *Recordset Operand*. The operator shall be applied to every record of the input *Recordset* and the *Scalar*.

- For **two *Recordsets***:
  - **Constraints**: Binary *Operators* can only be applied to two *Recordsets Operands* if they have:
    - Exactly the same *Key Components;* or
    - the *Key Components* of one *Recordset* (Reference *Recordset*) are a superset of the *Key Components* of the other *Recordset*.
  - **Behaviour**: Performs an inner join and the operator applies to the pairs of values resulting from performing an inner join.

Null treament

Treatment of null is specified for each operator

Broadly speaking, null is intended, and treated, as unknown

- *Unknown = 5 → Unknown*
- *Unknown and false → false*

Null evaluations are not considered errors

Standard behaviour can be modified

- By using the *default* clause
- By using the *nvl* operator

Interval arithmetics

Each operand can be defined as point or interval (in the selection)

For each operator, there is a specification of the calculation to apply when intervals are applied

Calculations are based on Eurofiling’s specification, although there are minor implementation differences

Aggregate operators

Aggregate operators perform operations on the measures of the operand recordset, calculating the required aggregated values for groups of records. The groups of records to be aggregated are specified through the grouping clause. If no grouping clause is used, the operation shall be calculated on all the records, resulting in a scalar.

In practice, aggregate operators take as input a recordset with a set of keys and records, and return another recordset with fewer keys and records, performing an aggregation operation.

Syntax

```
aggregateOperator (op {group by groupingId {, groupingId}*})
```

Operators enumeration

- sum
- avg
- count
- min_aggr
- max_aggr
- median

Aggregate operators: Group by

Suppose the following recordset:

RCP	r	c	f
ES	20	10	200
ES	30	10	300
PT	20	10	500
PT	30	10	600
ES	20	20	100
ES	30	20	400
PT	20	20	700
PT	30	20	800

Grouping by implies creating groups of records by the set of keys provided, so that the desired aggregate operation is performed for all the records of the group.

The output dataset will have the keys included in the *group by*, any other key will be dropped.

Group by RCP

RCP	r	c	f
ES	20	10	200
ES	30	10	300
PT	20	10	500
PT	30	10	600
ES	20	20	100
ES	30	20	400
PT	20	20	700
PT	30	20	800

Group by r

RCP	r	c	f
ES	20	10	200
ES	30	10	300
PT	20	10	500
PT	30	10	600
ES	20	20	100
ES	30	20	400
PT	20	20	700
PT	30	20	800

Group by r, c

RCP	r	c	f
ES	20	10	200
ES	30	10	300
PT	20	10	500
PT	30	10	600
ES	20	20	100
ES	30	20	400
PT	20	20	700
PT	30	20	800

Aggregate operator: Sum example

```
sum({t1,r20-30, c10-20}
group by RCP)
```

RCP	r	c	f
ES	20	10	200
ES	30	10	300
PT	20	10	500
PT	30	10	600
ES	20	20	100
ES	30	20	400
PT	20	20	700
PT	30	20	800

RCP	f
ES	1000
PT	2600

```
sum({t1,r20-30, c10-20}
group by r)
```

RCP	r	c	f
ES	20	10	200
ES	30	10	300
PT	20	10	500
PT	30	10	600
ES	20	20	100
ES	30	20	400
PT	20	20	700
PT	30	20	800

r	f
20	1500
30	2100

```
sum({t1,r20-30, c10-20}
group by r, c)
```

RCP	r	c	f
ES	20	10	200
ES	30	10	300
PT	20	10	500
PT	30	10	600
ES	20	20	100
ES	30	20	400
PT	20	20	700
PT	30	20	800

r	c	f
20	10	700
30	10	900
20	20	800
30	20	1200

```
sum({t1,r20-30, c10-20})
```

f
3600

Max(Min) vs max_aggr(min_aggr)

For max and min, it should be noted that there are two different versions of the operator: The binary and the aggregate.

The **binary version**, takes as input two (or more) operands, and behaves like any binary operator.

The **aggregate version**, takes as input one operand, and, optionally, a group by clause, and behaves like any aggregate operator.

```
max(2, 3, 4)
```

F
4

```
max_aggr({t1,r20-30,
c10-20} group by RCP)
```

RCP	f
ES	400
PT	800

Filter example

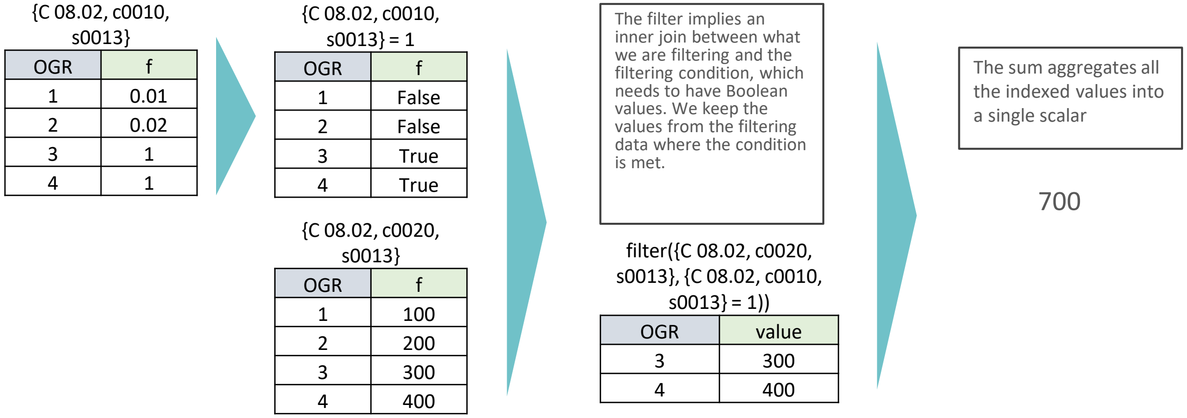
The filter operator takes as input two recordsets. The first one is the one we want to filter, and the second is the filtering criterion.

Binary operators' constraint apply in what regards the keys of the input recordsets.

In the current version of the validations, where is used only inside the aggregate operators.

```
sum(where({C 08.02,c0010,s0013}=1) {C 08.02,c0020, s0013})
```

```
sum(
   filter(
      {C 08.02, c0020, s0013},
      {C 08.02, c0010, s0013} = 1)
)
```



{C 08.02, c0010, s0013}

OGR	f
1	0.01
2	0.02
3	1
4	1

{C 08.02, c0010, s0013} = 1

OGR	f
1	False
2	False
3	True
4	True

{C 08.02, c0020, s0013}

OGR	f
1	100
2	200
3	300
4	400

The filter implies an inner join between what we are filtering and the filtering condition, which needs to have Boolean values. We keep the values from the filtering data where the condition is met.

filter({C 08.02, c0020, s0013}, {C 08.02, c0010, s0013} = 1))

OGR	value
3	300
4	400

The sum aggregates all the indexed values into a single scalar

700

Where Example

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

The where operator serves to filter by the values of one of the key components of the recordset

```
{tF_20.05, r0020-0030, c0010}[where
RCP = “ES”]
```

RCP	r	f
ES	0020	200
ES	0030	300

time_shift

The time_shift operator serves to change a date component by shifting a date component by a number of periods.

```
time_shift(op, period, numberPeriods, {var})
```

{tT1, r010-020, c010-020}

refPeriod	r	c	f
2022Q3	10	10	100
2022Q3	10	20	200
2022Q3	20	10	300
2022Q3	20	20	400
2022Q4	10	10	500
2022Q4	10	20	600
2022Q4	20	10	700
2022Q4	20	20	800

time_shift({tT1, r010-020, c010-020}, Q, 1, refPeriod)

refPeriod	r	c	f
2022Q4	10	10	100
2022Q4	10	20	200
2022Q4	20	10	300
2022Q4	20	20	400
2023Q1	10	10	500
2023Q1	10	20	600
2023Q1	20	10	700
2023Q1	20	20	800

time_shift({tT1, r010-020, c010-020}, Q, -1, refPeriod)

refPeriod	r	c	f
2022Q2	10	10	100
2022Q2	10	20	200
2022Q2	20	10	300
2022Q2	20	20	400
2022Q3	10	10	500
2022Q3	10	20	600
2022Q3	20	10	700
2022Q3	20	20	800

In practice, it is used for comparing information in reference dates

```
{r0010} = {r0210} t-1
```

```
with {tF_46_00, c*}: {r0010} =
time_shift({r0210}, A, 1, refPeriod})
```

Aggregation exercise

```
with {s*, default: null, interval: true}:
    {tC_08.01.a, r0070, c0240} * sum ({tC_08.02, c0140} group by s) = sum ({tC_08.02, c0240} * {tC_08.02, c0140})
```

{C 08.02, c0140}

s	OGR	f
0001	1	100
0001	2	200
0002	1	300
0002	2	400

{C 08.02, c0240}

s	OGR	f
0001	1	1
0001	2	2
0002	1	2
0002	2	3.75

{C 08.01.a, r0070, c0240}

s	f
0001	2
0002	3.71

sum({C 08.02, c0140} group by s)

s	f
0001	300
0002	700

{C 08.02, c0240} * {C 08.02, c0140}

s	OGR	f
0001	1	100
0001	2	400
0002	1	600
0002	2	1500

{C 08.01.a, r0070, c0240} * sum({C 08.02, c0140} group by s)

s	f
0001	600
0002	2600

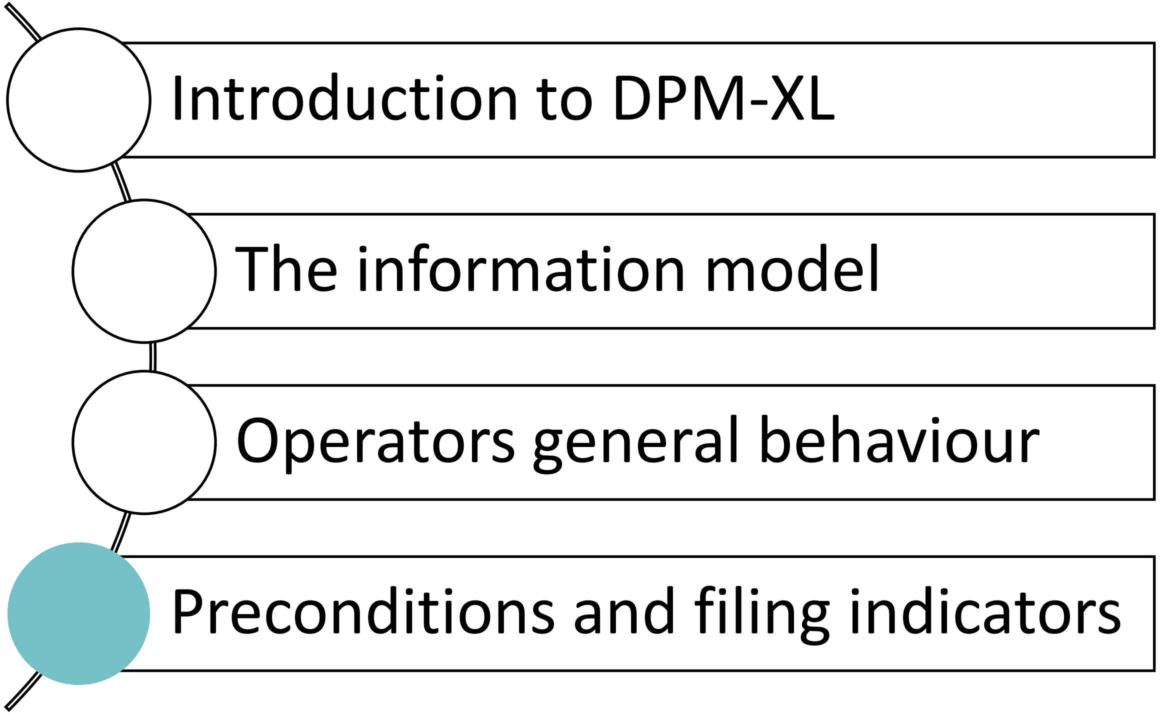
sum({C 08.02, c0240} * {C 08.02, c0140})

f
2600

{C 08.01.a, r0070, c0240} * sum({C 08.02, c0140}, (sheet)) = sum({C 08.02, c0240} * {C 08.02, c0140})

s	f
0001	False
0002	True

Outline



- Introduction to DPM-XL
- The information model
- Operators general behaviour
- Preconditions and filing indicators

Preconditions

Any DPM-XL validation may have one or no precondition.

Preconditions are **normal DPM-XL operations**.

Condition: need to yield as a result a **Boolean scalar**.

Preconditions are **evaluated first**. If the result is false, the validation is not executed

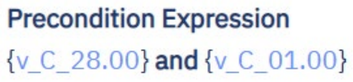
Filing indicators

Filing indicators are explicitly defined in the DPM.

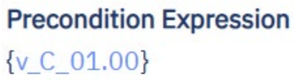
They are Boolean variables with a specific code.

DPM-XL allows selecting variables (by using *v_* notation in the selection operator).

Most preconditions operate with filing indicators, using logical operators.



Precondition Expression
{v_C_28.00} and {v_C_01.00}



Precondition Expression
{v_C_01.00}

Thank you!

eba European Banking Authority

Floor 24-27, Tour Europlaza
20 Avenue André Prothin
92400 Courbevoie, France

Tel: +33 1 86 52 70 00
E-mail: info@eba.europa.eu

https://eba.europa.eu/