byron

Byron Informatik AG
Lohweg 6
CH-4054 Basel
Tel. +41 (0)61 690 96 00

byron@byron.ch
www.byron.ch

Byron/BIS – Technical product information
Filter Navigation Description

Table of contents

1 Syntax

The filter and navigation description is a formal language to describe objects in Byron/BIS.

```
FilterNavigation   =  Statements
Statements     =    { Statement }
Statement      =    Startdef | Setoperation | CountStatement | Classfilter |
                    Sort | CompareFilter | Valuefilter | References | Loop |
                    Navigation | SaveAs | Recall | Duplicates | GroupObjects
                    | PickObjects | ConditionStmnt | BlockStmnt | PassStmnt
                    | RightFilter | IF_Stmnt | CalcFilter | DefaultObjectNav
                    | LogStmnt |
                    CreateObject | DeleteObject | CopyObject | ChangeClass |
                    ModifyAttribute | SetDefaultValue | CopyAttribute |
                    ModifyAssociation | ExecuteAction | ExecuteNav |
                    SetDefContainer | SetParameter | Iterate | Break |
                    ModelOperation.
Startdef       =    StartInstances | StartValue | StartClass |
                    StartContainer |
                    StartPerson | StartUser | StartObject |
                    StartDefaultObject | StartMinMax.
StartInstances=     'START' 'INSTANCES' ['='] Class ['NOT' ({string |
                    ExternalValue })].
StartValue     =    'START'
                    ('VALUE' Attribute [IndexOp] Value [SimpleOp Value] |
                     'REFERENCES' ['NOT'] Association '(' Statements ')' )
                    {'ANDVALUE' Attribute [IndexOp] Value [SimpleOp Value] |
                     'ANDREFERENCES' ['NOT'] Association '(' Statements ')'
                    |
                     'ANDCLASS' ['!'] ['='] Class }
                    ['SKIP' Value] ['MAX' Value].
StartClass     =    'START' ('TYPE' | 'CLASS') Class.
StartContainer=     'START' 'CONTAINER' Class.
StartPersonUser     =   'START' 'PERSON'.
StartPersonUser     =   'START' 'USER'.
StartObject    =    'START' (String | ExternalValue).
StartDefaultObject =   'START' 'DEFAULTOBJECT' Class.
StartMinMax    =    'START' ('MIN' | 'MAX') Attribute [Value].
Setoperation   =    '[' union | intersection | difference | alternative ']'.
Union          =    Statements { 'OR' Statements }.
Intersection   =    Statements { 'AND' Statements }.
Difference     =    Statements { 'ANDNOT' Statements }.
Alternative    =    Statements { 'ELSIF' [SimpleOp integer] Statements }.
CountStatement=     'COUNT' '(' Statements ')' [ SimpleOp. IntegerValue].
Classfilter    =    ('TYPE' | 'CLASS') ['!'] ['='] Class.
Sort           =    'SORT' ['DISTINCT'] '(' AccordingAttrib
                    {AccordingAttrib} ')'.
```

Cont. next page

```
CompareFilter =    'COMPARE' ['FOR' (integer | 'ALL' | ExternalValue)]
                   (Expression | RefAttribute) '(' {Statement} ')'
                   [SimpleOp] [RefAttribute] '('  {Statement} ')'.
Valuefilter   =    'VALUE' (AttributeE | RefAttribute)
                   ['NOT' | IndexOp] Value.
References    =    'REFERENCES' ['NOT'] Association '(' Statements ')'.
Loop          =    'LOOP' ['FOREACH'] '(' Statements ')' [IntegerValue].
Navigation    =    'VIA' Association.
SaveAs        =    'SAVEAS' ['GLOBAL' ['FLAT'] | 'TRANS' ] ident ['PROMPT'
                   string | Expression].
Recall        =    'RECALL ident.
Duplicates    =    'DUPLICATES' Attribute.
GroupObjects  =    'GROUPBY'  Attribute '(' Statements ')'.
PickObjects   =    'PICK' ('=' IntegerValue | IntegerValue [IntegerValue]).
BlockStmnt    =    'BLOCK' [Expression].
PassStmnt     =    'PASS' [Expression].
ConditionStmnt=    'CONDITION' Expression.
RightFilter   =    GroupRightFilter | SchemaRightFilter |
                   ObjectRightFilter.
GroupRightFilter   =   'CHECKRIGHT' Attribute.
SchemaRightFilter  =   'CHECKRIGHT' (Attribute | Association) ['='] ('READ'
                   | 'WRITE').
ObjectRightFilter  =   'CHECKRIGHT' '(' { 'READ' | 'WRITE' | 'CREATE' |
                   'DELETE' } ')'.
IF_Stmnt      =    'IF' Expression '(' Statements ['ELSE' Statements] ')'.
CalcFilter    =    'CALCULATED' ['NOT']  Attribute.
DefaultObjectNav   =   'DEFAULTOBJECT'.
LogStmnt      =    'LOG' StringValue ['DEEP'] [integer].
CreateObject  =    'CREATE' [integer] Class ['(' Statements ')'].
DeleteObject  =    'DELETE'.
CopyObject    =    'COPYOBJECT' '(' Statements ')' ['FLAT' | 'DEEP'].
ChangeClass   =    'CHANGECLASS' Class.
ModifyAttribute    =   'MODIFY' Attribute Value.
SetDefaultValue    =   'SETDEFAULTVALUE' Attribute.
CopyAttribute =    'COPY' Attribute '(' Statements ')' Attribute.
ModifyAssociation  =   'BIND' Association '(' Statements ')' ['ADD' |
                   'REBIND'].
ExecuteAction =    'EXECUTEACTION' StringValue.
ExecuteNav    =    'EXECUTENAV' ['MODIFY'] StringValue.
SetDefContainer    =   'SETDEFAULTCONTAINER' Class.
SetParameter  =    'SETPARAMETER' ['(' Statements ')'] StringValue
                   StringValue.
Iterate       =    'ITERATE' ['='] Class [Value] '(' Statements ')'.
```

Cont. next page

```
Break         =    'BREAK' | 'BREAKLOOP'.
```

```
ModelOperation=    'MODEL' ( 'DELETE' StringValue [StringValue] |
                             'CREATESYNONYM' StringValue StringValue |
                             'RENAME' StringValue StringValue |
                             'DELETEMETHOD' StringValue StringValue ['SET'] |
                             'SETDEFAULTVALUE' StringValue StringValue
                   [Value]).

SimpleOp       =   ('=' | '<' | '>' | '<=' | '>=' | '<>').
IndexOp        =   (SimpleOp | '~=' | '!~' | '^=' | '!^').
Value          =   [literal | ExternalValue | Expression] .   **)
StringValue    =   [String | ExternalValue | Expression] .   **)
IntegerValue   =   [integer | ExternalValue | Expression] .   **)
AccordingAttrib    =   ['+' | '-'] RefAttribute.
RefAttribute   =   [quote]ident {'%' ident}[quote].   *)
Class          =   [quote] ident [quote] | ExternalValue.   *)
Association    =   ([quote] ident [quote] | ExternalValue).
Attribute      =   ([quote] ident [quote] | ExternalValue).
AttributeE     =   ([quote] ident [quote] | ExternalValue | Expression).
ExternalValue  =   'DATAOF' ident ['PROMPT' string] | '$' ident '$'.
ident          =   letter {character}.
literal        =   integer | string | float | 'true' | 'false'.
String         =   quote {character} quote.
quote          =   '"' | '''.
Expression     =   String.   ***)
```

*) if an ident contains special characters like '%' or '@' the identifier must be quoted
invalid: anassoc%cls.aa@sds.aa
valid: 'anassoc%cls.aa@sds.aa'

**) How to express values for different data types is explained later in this document.

***) An expression starts with a "=" sign. Also see expression documentation

Comments

Single line comments begins with two "-" signs

Example

```
-- This is a single line comment
```

Multi line comments begins with a "{" sign and ends with a "}" sign.

Example

```
{ This is a multi line comment
  with 2 lines }
```

1.1 Statements

The input of FilterNavigation is a (list) object. This object can be produced in FilterNavigation itself by a Start statement. Each statement has a list of objects as input and produces a list of objects as output. Some statements (CLASS, VALUE, COUNT) are filters, that is: the output list contains all objects of the input list (never more), but only those objects, which fulfill the condition of the statement. With the Via Statement other objects can be reached (Navigation Statement). The result of a navigation statement does never contains duplicates. A sequence of filter statement is a logical AND of this conditions (result fulfills condition 1 AND condition 2). A logical OR is expressed as Union [OR OR …].

1.2 Start

To define a set of objects, use START. There are following mechanisms to do it:

1.2.1 START INSTANCES

Syntax: `'START' 'INSTANCES' ['='] Class ['NOT' ({string | ExternalValue })].`

INSTANCES takes all objects in the Database, which belong to this class, or subclass, if there is no '='.

`START INSTANCES 'Room'`	All objects of class or subclass of Room
`START INSTANCES = 'Room'`	All objects of class Room (no derived objects)

```
START INSTANCES 'Space' NOT ('Floor' 'Building')
```

All objects derived of the class Space, but no objects of class Floor and no objects of class Building.

1.2.2 START VALUE / START REFERENCES

Syntax:
```
'START'
('VALUE' Attribute [IndexOp] Value [SimpleOp Value] |
 'REFERENCES' ['NOT'] Association '(' Statements ')' )
{'ANDVALUE' Attribute [IndexOp] Value [SimpleOp Value] |
 'ANDREFERENCES' ['NOT'] Association '(' Statements ')' |
 'ANDCLASS' ['!'] ['='] Class }
['SKIP' Value] ['MAX' Value].
```

`ANDVALUE` extension is available in Byron/BIS **>= v5.0.0**

`REFERENCES`, `ANDREFERENCES` and `ANDCLASS` extension is available in Byron/BIS **>= v5.0.3**.

`START VALUE` supports the compare operator "**not match**" i.e. **'!~'** in Byron/BIS **>= v5.0.3**

`START VALUE` supports the compare operators "**exists**" i.e. **'^='** and "not exists" i.e. **'!^'** in Byron/BIS **>= v5.8.0**. "not exists" **must** always be used with an additional condition like `ANDCLASS` in order to avoid to large result sets. Otherwise, **no results** are returned, and a warning is written to the log.

START VALUE finds objects by an indexed attribute search. Default compare operation is ~= (Match). Unlike the standard VALUE-Filter (without START), an asterisk (*) is automatically added to the search-value if no wildcard is used in the search-value:

```
START VALUE bisB_Lastname 'M'
```

SAME AS

```
START VALUE bisB_Lastname 'M*'
```

BUT

```
START VALUE bisB_Lastname '*M'
```

DOES NOT AUTOMATICALLY ADD AN ASTERISK (*) AT THE END

To get objects by a value range use two comparisons:

```
START VALUE bisT_StartDateTime >= '1.7.2006' < '1.8.2006'
```

matches to all objects in time range.

To get not more than 5 results in a query (MAX was introduced in v4.9.0):

```
START VALUE bisT_StartDateTime >= '1.7.2006' < '1.8.2006' MAX 5
```

The ANDVALUE extension was introduced to support optimized search with SQL databases. The statement

```
START VALUE bisT_StartDateTime <= '5.7.2006'
ANDVALUE bisT_EndDateTime > '1.7.2006'
```

is equivalent to

```
[START VALUE bisT_StartDateTime <= '5.7.2006' AND
 START VALUE bisT_ EndDateTime > '1.7.2006']
```

The REFERENCES / ANDREFERENCES extension was introduced to support optimized search on indexed associations with SQL databases. The statement:

```
START VALUE bisb_SpaceId ~= 'B-*'
ANDREFERENCES bisC_RoomToSpaceUsageType (
    START VALUE "bisC_SpaceUsageNumber" = "2.1")
```

Searches all Rooms whose id starts with "B-" that have the usage type 2.1 (Office). **Note** that only Rooms are found, because other space objects do not have an association to usage type.
The statements that evaluate the referenced object are executed only once for the input set. Only the first object of the result is used as referenced object.

The ANDCLASS extension was introduced to support optimized search with SQL databases. It supports the same directives ('!', '=') as the class filter.

Only **one** of the attributes / associations used in START VALUE … ANDVALUE … ANDREFERENCES … needs to be indexed. All conditions with attributes that are not indexed are translated into VALUE statements (be careful when using the match-operator without an '*' at the end, because it behaves differently (see above).

Get 50 result entries and skip the first 50 results of a search **>= v5.5.4**

```
START VALUE bisT_StartDateTime >= '1.7.2006' < '1.8.2006' SKIP 50 MAX 5
```

Note:

- One of the attributes or associations (`bisB_Lastname`,`bisT_StartDateTime`, `bisC_RoomToSpaceUsageType`) has to be indexed.
  Indexed associations are only supported with **SQL databases**.
- Do never use = comparison for floats or dates
- `MAX 0` returns all results
- Only **one** object is used as referenced object when executing `ANDREFERENCES`.
- With SQL databases, do only use the indexed attributes / associations / classes in a statement, that really narrow the result set. A **bad** example is:
  ```
  START VALUE bisb_SpaceId ~= 'B-*' ANDCLASS Room.
  ```
- Do **not** use `ANDCLASS` with an object class that has a lot of subclasses, unless you use the directive `'='`.
- Looking for not existing values is **not** supported with one exception:
  `START REFERENCES Object_To_Parent (BLOCK)` works with SQL databases.
- If **`SKIP`** is used, **`MAX`** can only have the values 0, 50, 100, 250, 500, 1000, 2500 and 5000. All other values are rounded (e.g. 1-49 => 50, 10000 => 5000).
- If **`MAX`** is used with **`ANDVALUE`**, the used attribute has to be indexed (otherwise the result is similar to **`VALUE`** after use of **`MAX`**).
- With **SQL** databases, `START VALUE bisB_EMail" = "hirsch@byron.demo"` or `START VALUE bisB_EMail" = "hirsch@byron.demo "` find the same objects, because **spaces** at the end of the string are **ignored**.

Examples using `START REFERENCES`

Starting with Byron/BIS **v5.8.0**. `START REFERENCES` allows to query for (not) existance of an association. To avoid confusion and mistakes, some examples are given. Also see REFERENCES

- Find all objects without a parent.
  This query was already supported in Byron/BIS < v5.8.0
  ```
  START REFERENCES "Object_To_Parent" (BLOCK)
  ```
- Find all objects with a parent.
  This query was already supported in Byron/BIS < v5.8.0.
  ```
  START REFERENCES NOT "Object_To_Parent" (BLOCK)
  ANDCLASS "RootContainer"
  ```
  This query **must** always be used with an additional condition like `ANDCLASS` to avoid to large result sets. Otherwise, **no results** are returned, and a warning is written to the log.
- Find all Rooms without a usage type
  ```
  START REFERENCES "bisC_RoomToSpaceUsageType" (BLOCK)
  ANDCLASS "Room"
  ```
  This query **must** always be used with an additional condition like `ANDCLASS` to avoid to large result sets. Otherwise, **no results** are returned, and a warning is written to the log.
- Find all Rooms with a usage type
  ```
  START REFERENCES NOT "bisC_RoomToSpaceUsageType" (BLOCK)
  ```

Example – Iterate over a search result with `SKIP` – packet size is 100

```
SAVEAS "skip" "=0"
LOOP (
  LOG "Iterate"
  [
  OR
    START VALUE "BBI_DatenpaketNummer" ~= "#B3G=T*"
          SKIP DATAOF "skip" PROMPT "skip" MAX 100
    IF 'COUNT = 0' (
      BREAKLOOP
    )
  ]
  SAVEAS "skip" "=skip + 100"
) 10000
```

1.2.3 START CLASS

Syntax: **`'START' 'CLASS'`** `Class.`

CLASS (or obsolete TYPE) returns all “root objects” of the given class. A root object is an object that is not contained in any other object (has no parent). In a tree view, root objects are typically shown at the top level.

`START CLASS 'Space'`

returns the container of all space objects ('Root_Space')

`START CLASS 'Root'`

returns all “root objects” .

1.2.4 START CONTAINER

Syntax: **`'START' 'CONTAINER'`** `Class.`

CONTAINER returns the default container of the specified class.

`START CONTAINER 'Room'`

Returns the `START CLASS Space` Object

1.2.5 START PERSON

Syntax: **`'START' 'PERSON'.`**

PERSON returns the current person (or nothing). As default the current person is the person that is associated to the current Byron/BIS user.

1.2.6 START USER

Syntax: **`'START' 'USER'.`**

USER returns the current user (or nothing).

1.2.7 START Object Image String

Syntax: `'START' (String | ExternalValue).`

Start can define a set of one or more objects (available in Byron/BIS >= v5.0.3) if an object image string is given as shown in the examples below. Multiple objects are specified with al list of object images separated by a comma.

```
START "{4656B4CB-8E31-4080-8B8D-83F0F0CB0D2B}"

START " {4656B4CB-8E31-4080-8B8D-83F0F0CB0D2B},
        {B2CB9EA9-9CBB-4773-B7F6-CDD50D424ADD}"
```

1.2.8 START External Value

Syntax: `'START' ExternalValue.`

Depending on the context, FilterNavigation can be started with a set of objects stored in an external variable. Currently this is possible within the XDS application (extensions XMLInput, XMLOutput and GenerateBenchmarks). Note: Recall does this job too.

1.2.9 START DEFAULTOBJECT >= v4.2

Syntax: `'START' 'DEFAULTOBJECT' Class.`

Returns the default object of a class and all its subclasses. The default object is the object containing the class properties.

```
START DEFAULTOBJECT Space
```

1.2.10 START MIN or START MAX >= v5.0.3

Syntax: `'START' ('MIN' | 'MAX') Attribute [Value].`

Returns the objects that have the n largest (`MAX`) or smallest (`MIN`) values of an **indexed** attribute. The result number is given by `Value`. The default for `Value` is one (1).

```
START MAX bisP_OrderNo
```

1.3 Condition Statements

1.3.1 CLASS

Syntax: `('TYPE' | 'CLASS') ['!'] ['='] Class.`

The resulting objects are of the specified type (BIS class) or subclass. TYPE is an obsolete synonym of CLASS.

Examples

`CLASS Room`

rooms only, no floors or other objects. Objects of classes derived from Room (e.g. Lab) are in the result set too.

`CLASS = Room`

rooms only, no floors or other objects. Objects of classes derived from Room (e.g. Lab) are **not** in the result set.

`CLASS ! Room`

All object classes except room. Objects of classes derived from Room (e.g. Lab) are **not** in the result set.

`CLASS != Room`

All object classes except room. Objects of classes derived from Room (e.g. Lab) are in the result set too.

1.3.2 CONDITION <expression> >= v4.5

Syntax: `'CONDITION' Expression.`

CONDITION is a filter which **evaluates each object**. If the expression returns 0/false, the object is blocked, otherwise it is passed.

Example shows all forms changed at a 13.th

```
START INSTANCES bisB_FormObject
CONDITION "=DAY(ATTRIBUTE('bisB_ModificationDate'))=13"
```

1.3.3 BLOCK PASS

Syntax: `'BLOCK' [Expression].`
`'PASS' [Expression].`

BLOCK and PASS are filters which pass all objects or none. PASS is a 'null-statement', the effect is like 'CLASS ROOT' but better performed. BLOCK is like 'CLASS Space CLASS User', but better performed. Both statements may have an expression. In this case the statement is active if the expression results with a none 0 value or TRUE.

Note the expression is **evaluated only once** with the first object if there is any. This is important, if the expression contains an ATTRIBUTE function. In this case use a CONDITION statement.:

poor: `PASS '=ATTRIBUTE("name")="Meier" '`
you: `CONDITION '=ATTRIBUTE("name")="Meier" '`
or: `COUNT (VALUE name = "Meier") > 0`

Examples

```
BLOCK
```

This may be useful as last statement, after modification statements, to avoid a lot of objects in the result list in the search (to save time).

```
BLOCK '= a = "F2"'
```

If the Parameter 'a' has the value 'F2', no objects results from this statement, otherwise all input objects.

```
PASS
```

If a navigation is needed, which returns the input objects, this is perfect.

```
PASS '= HASLIC("_BISLayout")'
```

If the license "_BISLayout" exists in the database, all input objects are returned. This could be used to display an application with graphics only if there is a specific license.

1.3.4 VALUE

Syntax: `'VALUE' (Attribute | RefAttribute) ['NOT' | IndexOp] Value.`

The resulting objects fulfill the value condition. The default compare Operation is ~= (Match). A list of all compare operation is found at the end of this document. With DATAOF external values can be accessed, see Examples.

Examples

```
VALUE Name 'Meier*'
```

'Meier' and 'Meierhans' pass, but 'Meyer' does not

```
VALUE Name = 'Meier'
```

'Meier' pass, but 'Meierhans' does not

```
VALUE 'Object_To_Parent%Name' 'M*'
```

It is the name of the parent objects, which must be start with a 'M'. If the reference can contain more than one object (Object_To_Children), the result may be not what you expect (see COUNT statement)

```
VALUE Name = $nm$
```

obsolete form: Name is compared with the Value of Parameter nm see Notes

```
VALUE Name = DATAOF nm PROMPT "First Name"
```

Name is compared with the Value of Parameter "nm"; see Notes

```
START INSTANCES Room
VALUE "bisR_PersonCapacity" >
"=ATTRIBUTE('acl_NetGroundArea') / 4"
```

To get all rooms whose person capacity is larger than the area divided by 4

Notes:

Some compare operations are expressed different in a XML environment '<' is written as <

The compared value can be expressed as parameter. A parameter is recognized if the identifier starts with a '$' (obsolete form) or by the keyword DATAOF. Parameters are implementation

specific. When used in a search expression, the user is prompted for input; in XDS a parameter is a variable, within scripting it produces an OnParameter callback.

If the Attribute name contains special characters like % or @ use ''.

If the Attribute is a Boolean use = true / = false to check.
Example: `VALUE IsOk = true`

1.3.5 REFERENCES >= v5.0.3

Syntax: `'REFERENCES' ['NOT'] Association '(' Statements ')'.`

Evaluates for each of the objects in the input set, if it references the first object that was found with the statements in the brackets.

```
START VALUE bisb_SpaceId ~= 'B-*'
REFERENCES bisC_RoomToSpaceUsageType (
    START VALUE bisC_SpaceUsageNumber = '2.1'
)
```

Searches all Rooms whose id starts with "B-" that have the usage type 2.1 (Office). **Note** that only Rooms are found, because other space objects do not have an association to usage type.
The statements that evaluate the referenced object are executed only once for the input set. Only the first object of the result is used as referenced object.

When no referenced object is given, the filter accept objects, that do not have referenced objects. E.g.

```
START VALUE bisb_SpaceId ~= 'B-*'
REFERENCES bisC_RoomToSpaceUsageType (BLOCK)
```

lists all spaces whose id starts with "B-" that do not have an association to a usage type. This filter can be inverted with `NOT`.

See also START VALUE / START REFERENCES.

1.3.6 COUNT

Syntax: `'COUNT' '(' Statements ')' [ SimpleOp. IntegerValue].`

Count starts a 'subquery' for each object in the input list. The number of objects in the resulting list is compared with the specified number and compare operation. Default is > 0.

Examples

```
COUNT ( VIA Object_To_Children  VALUE Name 'M*')
```

all objects which has at least 1 child the name is starting with 'M'

```
COUNT ( VIA Object_To_Children
  CLASS Room
  [ VALUE Name '*WC*'
    OR VIA Room_To_Component CLASS bisP_Printer]
  ) > 0
```

all objects (floors) which has at least 1 rooms which contains a computer or which has a 'WC'.

Notes:

This is really a filter (condition) statement, even if it contains navigation to other objects, they are used to build an internal list, at the end the number of objects in this list is used to decide whether the original input object is removed or not.

Byron/BIS >= v5.0.3 COUNT optimizes the execution of the contained statements to avoid unnecessary statement execution.

The objects in the input set are iterated **backwards**.

Performance 1

If only the number of associated objects has to be counted, it is **much** better to use an expression with OBJCOUNT than the COUNT statement.

```
COUNT (VIA Object_To_Children) > 5
```

equals

```
CONDITION '=OBJCOUNT("Object_To_Children") > 5'
```

Performance 2

REFERENCES **(>= v5.0.3)** can be used as an **efficient** alternative to COUNT. Example:

```
COUNT (VIA bisC_RoomToSpaceUsageType VALUE bisC_SpaceUsageNumber = '2.1')
```

equals

```
REFERENCES bisC_RoomToSpaceUsageType (
    START VALUE bisC_SpaceUsageNumber = '2.1'
)
```

1.3.7 COMPARE

Syntax: `'COMPARE' ['FOR' (integer | 'ALL' | ExternalValue)] (Expression | RefAttribute) '(' {Statement} ')' [SimpleOp] [RefAttribute] '(' {Statement} ')' .`

With the compare statement, two attribute values of two objects may be compared. If the compare operation succeeds, the objects are passed to the result set.

Starting from the input set, for each input object, two object sets (A and B) are constructed with the statement sequences within '(' ')'. If a statement sequence is empty, the set contains the input object.

For each object in set A the value of Attribute1 is compared to the value of Attribute2 of each object in set B.

The compare operation succeeds, if either one, n or all objects of B meet the condition with at least one object of A. The number of objects in B that is needed for the operation to succeed can be specified by the FOR part of the statement. The default is 1.

If Attribute2 is not given, then Attribute2 = Attribute1.

Examples

Assume every person has the attributes "area" and "areaDemand". To find all persons whose areaDemand is smaller than their area the statement is:

```
START INSTANCES Person
COMPARE areaDemand () < area ()
```

or

```
START INSTANCES Person
COMPARE FOR 1 areaDemand () < area ()
```

Assume a person has a height and every room has a roomheight. To find all rooms with more than 2 persons, whose heights are larger than the roomheight min 0.2m the statement is:

```
START INSTANCES Room
COMPARE FOR 3 "=ATTRIBUTE('roomheight') - 0.2" () >=
              height (VIA Room_To_Component CLASS Person)
```

1.3.8 DUPLICATES

Syntax: `'DUPLICATES' Attribute.`

With the DUPLICATES statement, objects with a unique value of the attribute are filtered. Only objects that share the same value are passed.

Examples

Find Duplicates:

```
START INSTANCES Person DUPLICATES bisB_Firstname
```

Assume a list of 'Person' is before the DUPLICATES. The resulting list contains those which have a first name like another Person.
Incoming ('Jan', 'Yolanda', 'Vanessa', 'Jan') results in objects ('Jan', 'Jan')

1.3.9 CHECKRIGHT >= v4.3

Syntax:
```
'CHECKRIGHT' Attribute.
'CHECKRIGHT' (Attribute | Association) ['='] ('READ' | 'WRITE').
'CHECKRIGHT' '(' { 'READ' | 'WRITE' | 'CREATE' | 'DELETE' } ')'.
```

Checks A) a group right or B) a schema right or C) a object right.

A) Group right check has the form:

```
'CHECKRIGHT' Attribute.
```

A *GroupRightAttribut* may be bisB_GRightPropertySheet, bisB_GRightFilterNavUpdate…

Example

```
START INSTANCES bisP_Beamer CHECKRIGHT bisB_GRightGraphicPos
```

Gives all beamers, if the user may position components in the graphic

B) Schema right check has the form:

```
'CHECKRIGHT' (Attribute | Association) ['='] ('READ' | 'WRITE').
```

Note: the use of an association instead of an attribute is available from v5.4.8

Examples

```
START INSTANCES bisP_Beamer CHECKRIGHT Position WRITE
```

Gives all beamers, if the user has write access to the attribute ‘position’. This check includes the parent classes, this means: If the user has write access at the beamer, but not for `bisP_OfficeEquipment` (parent class of `bisP_Beamer`) this results to ‘he has no write access.

Examples

```
START INSTANCES bisP_Beamer CHECKRIGHT Position = WRITE
```

Gives all beamers, if the user has write access to the attribute 'position'. This check includes not the parent class.

C) Object right check >= v4.7.1 has the form:

'CHECKRIGHT' '(' { **'READ'** | **'WRITE'** | **'CREATE'** | **'DELETE'** } **')'.**

Examples

```
START INSTANCES bisP_Beamer CHECKRIGHT ( WRITE DELETE )
```

Only beamers the user has the right for write and delete pass to the end.

1.3.10 IF ELSE >= v4.7.10

Syntax: **'IF'** Expression **'('** Statements [**'ELSE'** Statements] **')'.**

Evaluates the condition, expression if this returns TRUE or not 0 then the statements before the ELSE are executed, otherwise the statements after the ELSE. The ELSE part is optional.

The condition is an expression where variables can be referenced.

Example

```
IF 'nm <> "-"' ( VALUE Name '=nm + *')
```

No value filter is made if nm = '-'

```
IF 'REGEXP(nm, "^B.*")' (VALUE Name '*_*' ELSE VALUE Name '*BB*')
```

1.3.11 CALCULATED >= v4.9.8

Syntax: **'CALCULATED'** ['**NOT**'] Attribute.

This is a 'is calculated' filter. Property-name may be an attribute name or an association name. If NOT is specified, all object where the property is not calculated are passed.

1.4 Navigation Statements

1.4.1 VIA

Syntax: `'VIA' Association.`

The resulting objects are of the objects, which can be reached by the specified reference (association). The original objects are removed; they are not in the resulting list.

Examples

VIA Object_To_Children
All children

```
VIA Object_To_Children VIA Object_To_Parent
```

Removes all objects without children. This two statements are equivalent to
`COUNT (VIA Object_To_Children)`

Notes:

Duplicates are always removed

1.4.2 LOOP

Syntax: `'LOOP' ['FOREACH'] '(' Statements ')' [IntegerValue].`

This statement executes the statements within '()' n times, n is the specified number. If there is no number specified, or the number is negative, the statements are executed as many times as the resulting list changes (grows). If the FOREACH directive is given, the statements are executed for each object individually. Otherwise the statements are executed with all objects together.

Examples

```
START INSTANCES Building
```

```
LOOP ( VIA Object_To_Children )
```

The input list is expanded to the whole tree of each objects. After the LOOP statement the resulting list contains all floors and rooms but no buildings (the input objects are not copied to the resulting list).

```
LOOP ( VIA Object_To_Children ) CLASS Room
```

If the input are Rooms, floors or buildings, the result are rooms only, which are in the input floors buildings etc.

```
LOOP ( VIA Object_To_Children ) 2
```

Is equivalent to
`VIA Object_To_Children VIA Object_To_Children`

To get the transitive for an association use

```
[ LOOP ( VIA Object_To_Children ) OR ]
```

The union (`[ OR ]`) has the effect of copying the input list to the result.

NB: Duplicates are always removed

1.4.3 Union

Syntax: `'[' Statements { 'OR' Statements }. ']'.`

This statement is built by [ ]. Within this [ ] each group is separated by an OR. The resulting list contains the union (all) objects of each group. Duplicates are always removed.

Examples

```
[ CLASS Room OR CLASS Floor ]
```

The input list is reduced to all Rooms and Floors

```
[VIA Room_To_Component CLASS Computer OR VALUE Name = 'A*']
```

The resulting list contains all Computers of the input rooms and all objects which the Name starts with 'A'.

```
[ VIA Room_To_Component CLASS Computer
  VIA Room_Of_Component OR VALUE Name = 'A*' ]
```

All rooms which the Name starts with 'A' and all rooms which contains at least one computer

1.4.4 Intersection

Syntax: `'[' Statements { 'AND' Statements }. ']'.`

This statement is built by [ ]. Within this [ ] each group is separated by an AND. The resulting list contains the intersection of each group result.

Examples

1.4.5 Difference

Syntax: `'[' Statements { 'ANDNOT' Statements }. ']'.`

This statement is built by [ ]. Within this [ ] each group is separated by an ANDNOT. The resulting list contains the result of the first group minus the result of each following group.

Examples

```
[  VALUE KST_Kosten_Nummer ~= "2*" ANDNOT
   VALUE KST_Kosten_Nummer ~= "23*" ANDNOT
   VALUE KST_Kosten_Nummer ~= "27*"]
```

All objects with KST_Kosten_Nummer beginning with 2, but not objects with KST_Kosten_Nummer beginning with "23" or "27"

1.4.6 Alternative

Syntax: `'[' Statements { 'ELSIF' [SimpleOp integer] Statements }. ']'.`

This statement is built by [ ]. Within this [] each group is separated by an ELSIF. The groups of statements are executed from left to right until the condition after ELSIF matches not or the last group is executed. The result of the alternative is the result of the last executed group.

The default condition is "= 0"

Examples

```
[  VALUE KST_Kosten_Nummer ~= "27*" ELSIF = 0
   VALUE KST_Kosten_Nummer ~= "23*" ELSIF < 10
   VALUE KST_Kosten_Nummer ~= "24*"]
```

Select all object with KST_Kosten_Nummer beginning with "27", if there are none (result count = 0) then select all object with KST_Kosten_Nummer beginning with "23", if there are less then 10, select all object with KST_Kosten_Nummer beginning with "24".

1.4.7 PICK index count

Syntax: `'PICK' IntegerValue [IntegerValue].`

With this statement a subset of the objects in the input set can be selected by index. The first parameter indicates the index where the selection is to start (first = 0).

The second parameter indicates number of objects (default = 1). If the first parameter is less than 0 the index calculated from the end.

PICK is often used in conjunction with SORT or GROUPBY.

Example:

```
START INSTANCES "bisO_Order"
SORT (-bisO_Deadline)
PICK 0 9
```

Gets the 9 latest orders in the database.

Example
A, B, C, D, E -> **PICK** index count -> result

index	count	Result
0	-	A
1	2	B, C
-1	1	E
-3	2	C, D
3	9	D, E

Example:

```
PICK '=INT(RANDOM(COUNT))' 1
```

This returns a random object from the input set. **>= v4.10.2**

1.4.8 PICK = count >= v4.9.3

Syntax: `'PICK' '=' IntegerValue.`

If there are exactly count objects, all pass, otherwise all objects are blocked. Default of count is 1, it may be an expression.

Example:

```
START VALUE bisP_OrderState = 1
PICK = 1
```

Gets the order with state 1, if there is exactly one. If there are two or more, no object returns.

1.4.9 DEFAULTOBJECT >= v4.2

Syntax: `'DEFAULTOBJECT'.`

Navigates from an object to its default object. (see also: START DEFAULTOBJECT)

Example (selects all default objects of bisB_Container, that have "$Name$" in its Object-Display):

```
START INSTANCES bisB_Container
DEFAULTOBJECT
VALUE Object_Display "*$Name$*"
```

1.4.10 ITERATE >= v5.0.3

Syntax: `'ITERATE' ['='] Class [Value] '(' Statements ')'.`

Executes the statements in the brackets for all objects of an object class and all of its subclasses. If the directive `'='` is given, no objects of subclasses are iterated. The iteration is done in packets whose size is given by `Value`. The default is 500.

The iteration can be stopped using the BREAK statement.

`ITERATE` ignores the input set and has no output set. Use `SAVEAS` and `RECALL` to collect objects.

Note:

Creation and deletion of objects of the iterated class is **not** supported.

Examples:

```
BLOCK
SAVEAS "collected"
ITERATE "Space" 500 (
  [VALUE " Obsolete_Marker " = 1  OR  RECALL "collected"]
  SAVEAS "collected"
  IF 'COUNT > 1000' (BREAK)
)
RECALL "collected"
DELETE
```

1.5 Miscellaneous Statements

1.5.1 SAVEAS >= v4.1

Syntax: `'SAVEAS' ['GLOBAL' ['FLAT'] | 'TRANS'] ident ['PROMPT' string | Expression].`

There are 3 forms of this statement:

1. Save Objects

```
SAVEAS mydeer
```

The current (input) objects are stored under the name 'mydeer'. They can be recalled later by `RECALL mydeer.`

2. Save (Object Attribute) Value

```
SAVEAS myval "=ATTRIBUTE('Name') "
```

The expression is evaluated for the (first) object. The resulting value is stored in parameter (here 'myval'). This value can be used by a `DATAOF myval` for example in a VAULE statement.

3. Save a Value the user can enter >= v4.2

```
SAVEAS myval PROMPT 'Bezeichner, bzw. einen Teil davon'
```

The user is prompted for a value. This value can be used later for example in an expression: `"='*' + myval + '*' ".`

If the keyword `'GLOBAL'` (>= v4.10.5) is given, the objects or value is stored to global storage as well.

The scope of "global storage in BisWeb is the web session. There is no way to pass values from one web session to another.

The scope of "global storage" in the Byron/BIS executable is either "the tool" when using ToolKonfiguration or the Byron/BIS process for Filternavigations without scope (e.g. method language, coloring or tree search):

- RECALL of a value stored in tool A does **not** return a value in tool B.
- RECALL of a value stored in tool A does return a value in a Filternavigation without scope.
- RECALL of a value stored in a Filternavigation without scope does return a value in tool A or tool B.

When the Keyword `'FLAT'` (>= v5.1.1) is used, the values are always stored without a scope and therefore always accessible.

```
SAVEAS GLOBAL FLAT "CurrentOffer"
```

GLOBAL storage cannot be used with `'PROMPT'`.

If the keyword `'TRANS'` (>= v4.11.3) is given, the **string** value is stored to transaction local storage **only**.
N.B.:Transaction local storage is cleared after a transaction finishes.

1.5.2 RECALL >= v4.1

Syntax: `'RECALL ident.`

Saved objects can be reused by the RECALL statement

Example simple Recall

```
RECALL mydeer
```

Example search floor which 90% of BGF is greater then BGF

```
START INSTANCES Floor
 COMPARE "=ATTRIBUTE('bisB_GrossGroundArea') * 0.9" ()
bisB_CacheNGF ()
```

Example: Search newest and 4 weeks older, but at least after 1.1.2006

```
START VALUE bisT_StartDateTime >= '1.1.2006'
SORT (- bisT_StartDateTime)
SAVEAS m1
PICK 0 SAVEAS m2 "=ATTRIBUTE('bisT_StartDateTime') - 28"
RECALL m1
VALUE bisT_StartDateTime >= DATAOF m2
```

1.5.3 SORT

Syntax: `'SORT' ['DISTINCT'] '(' AccordingAttrib {AccordingAttrib} ')'.`
`AccordingAttrib = ['+' | '-'] RefAttribute.`

In some application, the order of the objects may be important, in this cases the SORT statement can be used. An other usage is to get the highest or lowest, you can combine SORT with PICK 0. To eliminate objects with the same attribute values, the sort statement can be used with the keyword DISTINCT

Examples

```
SORT (+ bisO_OrderStatus - bisO_Deadline)
```

sorted with ascending Order-State and descending deadline

```
SORT DISTINCT (+ bisO_OrderStatus + name)
```

after this statement, all objects have a different name Order-Sate value (not the same name or not the same Order-State).
Note: It is random, which objects pass and which objects are eliminated (if the objects have the same attribute combination).

1.5.4 GROUPBY

Syntax: `'GROUPBY' Attribute '(' Statements ')'.`

This statement builds groups from the objects in the input set and executes the *GroupExec* statements for each group. The groups are built based on *GroupValue* – an attribute or expression of the objects in the input set. The output set is built from the output sets of the *GroupExec*.

If *GroupValue* equals ““, “bisB_ObjectImage” is used. Input objects that do not have a *GroupValue* are grouped in a separate group.

Examples:

```
START INSTANCES "Room"
GROUPBY "demo_FloorCovering" (SORT (-acl_NetGroundArea) PICK 0 1)
```

Returns the smallest room for each floor covering.

```
START INSTANCES "Room"
GROUPBY "KST_einheit%Object_Display_Kontext"
  (SORT (+acl_NetGroundArea) PICK 0 1)
```

Returns the largest room for each cost center

```
START INSTANCES "Room"
GROUPBY "=SUBSTR(ATTRIBUTE('demo_SpacePath'), 5, 1)" (PICK 0 1)
```

Returns two rooms for each floor number. The floor number is identified by the fifth and sixth character of the room number.

1.5.5 LOG >= v4.9.3

Syntax: `'LOG' StringValue ['DEEP'] [integer].`

Writes log messages depending on the BIS_DEBUG_LEVEL value. The log level is set as second parameter.

If `DEEP` is specified each object is written to the Log.

The log level is an integer that can be specified as follows:

level	
4	debug (default)
3	information
2	warning
1	error

Example 1:

```
LOG "point C" 3
```

Example 2:

```
LOG "point D" DEEP
```

Example 3:

```
START CLASS Doc_Root_Container
SAVEAS v "=ATTRIBUTE('Object_Display_Typed')"
```

```
VIA Object_To_Children
LOG "='contents of ' + v" DEEP 3
```

1.5.6 BREAK >= v5.0.3

Syntax: **'BREAK'.**

When used within the ITERATE statement, the object iteration is stopped and the execution is continued after ITERATE.

When used somewhere else, the execution of the Filternavigation is stopped and the result set is empty.

Examples – see ITERATE >= v5.0.3

1.5.7 BREAKLOOP >= v5.5.4

Syntax: **'BREAKLOOP'.**

When used within the LOOP statement, the iteration is stopped and the execution is continued after LOOP.

When used within the ITERATE statement, see BREAK >= v5.0.3.

When used somewhere else, the execution of the Filternavigation is stopped and the result set is empty.

Example – see START VALUE / START REFERENCES

1.6 Modification Statements

The statements documented in this chapter need an update transaction to execute properly.

1.6.1 CREATE

Syntax: **'CREATE'** [integer] Class ['**(**' Statements '**)**'].

```
CREATE [ <times> ] <class> '(' <statements to find the container> ')'
CREATE <class> '(' <statements to find the container> ')'      before v4.4.0
```

`times`: integer; number of objects created, if not specified, it is the number of objects in the input set.
`class`: string; the class name, may be a parameter (DATAOF)

If no times is defined, for each object in the input set a new object of the given class is created. Otherwise times specifies the number of created objects. If there is a resulting object at the end of '('..')', this is the container of the new object, otherwise it is the default container of the class. With `DATAOF` an external value can be used for the class name.

Examples

```
START INSTANCES Person CREATE DATAOF cnm PROMPT "Enter class name:"
```

Create an object for each person in the database. The class of the created objects must be entered by the user.

```
START INSTANCES Room
VALUE "demo_NumberOfPersons" > 2
LOOP FOREACH
  (CREATE "demo_Copier"
   VIA Object_To_Parent) "=ATTRIBUTE('demo_NumberOfPersons')/2"
```

For every room in the database that has more than two persons create a copier. The number of objects created is calculated by an expression (e.g. one copier, if the room has two persons).

Create one Person in default container or under input object:
****)** if there is one

```
CREATE 1 Person ()      MODIFY bisB_Lastname 'Meier' // default container
CREATE 1 Person         MODIFY bisB_Lastname 'Meier' // default container
CREATE 1 Person (PASS)  MODIFY bisB_Lastname 'Meier' // under input **)
CREATE 1 Person (BLOCK) MODIFY bisB_Lastname 'Meier' // default container
```

1.6.2 DELETE

Syntax: **'DELETE'.**

Deletes the objects in the input set.

Examples

```
START INSTANCES Room
VIA Object_To_Children CLASS "demo_Copier"
DELETE
```

Delete the objects created by the second example for CREATE.

1.6.3 COPYOBJECT

Syntax: **'COPYOBJECT'** **'('** Statements **')'** [**'FLAT'** | **'DEEP'**].

Copies the objects in the input set to the new parent given by the statements in the "()". If DEEP is given, contained objects of the objects in the input set are copied too (Default is FLAT).

Examples

```
START VALUE bisB_SpaceId = 'B01-18'
SAVEAS geschoss
COPYOBJECT (RECALL geschoss VIA Object_To_Parent) DEEP
MODIFY bisB_SpaceId 'B01-19'
```

1.6.4 CHANGECLASS

Syntax: **'CHANGECLASS'** Class.

Changes the object class of the objects in the input set. With DATAOF an external value can be used for the class name.

Examples

```
START INSTANCES Room
CLASS = Room --subclasses of Room ignored
VALUE demo_DeletionDate ^= "exists"
CHANGECLASS demo_RoomDeleted
```

Changes the object class of all rooms to demo_RoomDeleted if a deletion date exists. Objects whose classes are derived from Room are ignored.

1.6.5 MODIFY Attribute

Syntax: **'MODIFY'** Attribute Value.

For each object in the input set, an attribute is modified. With DATAOF external values can be accessed. If an expression is used for setting the new value, you can access to other attributes to use these as input values. Expression is evaluated for each object. This makes an important difference if you use the functions RANDOM or ATTRIBUTE.

Example 1: Changes the name of the copier to a value that is prompted to the user (if FilterNavigation is executed in the object search context).

```
…
CLASS demo_Copier
MODIFY Name DATAOF nm PROMPT "Enter copiers name"
```

Example 2: Creates an internal identifier constructed of two other attributes.

```
…
CLASS KST_Kostenstelle
MODIFY demo_internalID "=ATTRIBUTE('demo_Division')+ '.'
                       +ATTRIBUTE('KST_Kostennummer')"
```

Example 3: Removes the attribute demo_DeletionDate from all rooms.

```
START INSTANCES Room
CLASS = Room -subclasses of Room ignored
MODIFY demo_DeletionDate "=NULL"
```

Example 4: Modifies the first name of a person in 4 different variants.

Variant 1:

```
START VALUE "bisB_Lastname" 'M*' PICK 0 1
MODIFY 'bisB_Firstname' 'Hans'
```

Variant 2:

```
START VALUE "bisB_Lastname" 'M*' PICK 0 1
SAVEAS vVal  'Hans'
MODIFY 'bisB_Firstname' DATAOF vVal
```

Variant 3:

```
START VALUE "bisB_Lastname" 'M*' PICK 0 1
SAVEAS vAttr 'bisB_Firstname'
SAVEAS vVal  'Hansli'
MODIFY DATAOF vAttr DATAOF vVal
```

Variant 4:

```
START VALUE "bisB_Lastname" 'M*' PICK 0 1
SAVEAS vAttr 'bisB_Firstname'
SAVEAS vVal  'Hans'
MODIFY DATAOF vAttr '=vVal + ''li'' '
```

1.6.6 SETPARAMETER >= v4.7.1

Syntax: `'SETPARAMETER' ['(' Statements ')'] StringValue StringValue.`

Modifies a global parameter of the database.
SETPARAMETER name value
name and value can be a parameter (DATAOF) or an expression.

Examples

```
SETPARAMETER ModelVersion "=FORMAT(NOW, 'yyyymmdd')"
```

User parameters can be modified, if statements are given within '(' ')' (>= v4.10.6). The statements must deliver objects of the class "User". **NB**: user parameters are shown in the Byron/BIS user interface **only** if they start with `"P_"`.

Examples

```
SETPARAMETER (START USER) "P_PlanOutDir" DATAOF "Destination directory"
```

Global and user parameters can be deleted by setting a null value (>= v4.10.6).

Examples

```
SETPARAMETER "ModelVersion" "=NULL"
```

1.6.7 SETDEFAULTVALUE of Attributes

Syntax: `'SETDEFAULTVALUE' Attribute.`

For each object in the input set, the default value of the specified attribute is retrieved. Can be used to reset some values after changing the database schema.

Examples

```
…
CLASS demo_Copier
SETDEFAULTVALUE demo_CalculatedPrice
```

NB: The CREATE statement already sets the attribute values to their defaults.

1.6.8 SETDEFAULTCONTAINER of Classes

Syntax: `'SETDEFAULTCONTAINER' Class.`

Sets the default container of `<class>` to the object in the input set.

~~1.6.9 RESETSERIALNUMBERS Attribute~~ >= v5.8.1

Syntax: `'RESETSERIALNUMBERS' Attribute.`

Resets the internal state of the CheckoutNumber. This is the next CheckoutNumber of this Attribut will get a 0.

ab v5.8.1 nicht mehr verfügbar

1.6.10 COPY Attributes

Syntax: `'COPY' SourceAttribute '(' Statements ')' TargetAttribute.`

For each object in the input set one or all attributes from the first object found by the statement sequence given in parentheses are copied to the object of the input set.
A "*" instead of an attribute name specifies that all attributes marked by the copy flag are copied.

Examples

```
START VALUE  "bisB_SpaceId" = 'Z-M08-03' VIA "Object_To_Children"
LOOP FOREACH ( SAVEAS s SAVEAS sn  "=ATTRIBUTE('Name')"
  START VALUE bisB_SpaceId = "=sn"  SAVEAS t
  RECALL t COPY bisB_AreaGeometrie (RECALL s) bisB_AreaGeometrie
)
```

Copies the geometry of each room in floor `z-M08-03` to the corresponding room with the `bisB_SpaceId` of the source room name.
e.g. if floor `'Z-M08-03'` contains a room named `'B-A20-01-02'` then the Geometry of this room is copied to the room with `bisB_SpaceId = 'B-A20-01-02'`

```
…
CLASS Room
COPY "*" (START VALUE demo_RoomTemplateID = "RT07") "*"
```

Copies all attributes of the object found with the indexed attribute `demo_RoomTemplateID` = RT07 to the objects of a set of rooms.

```
…
COPY "ORG_Nachname" () "bisB_Lastname"
```

Copies the value of the attribute `ORG_Nachname` to the attribute `bisB_Lastname`.

1.6.11 BIND - Modify Association

Syntax: `'BIND' Association '(' Statements ')' ['ADD' | 'REBIND'].`

Creates an association from each object of the input set to the objects found by the statement sequence given in parentheses. The Association can be set (specify `REBIND`) or modified – objects are added (specify `ADD`). The default is `ADD`.

Examples

```
…
CLASS KST_Kostenstelle
BIND demo_KSTtoAbteilung (BLOCK) REBIND
```

Removes all objects from the association `demo_KSTtoAbteilung` of each `KST_Kostenstelle`. The filter `BLOCK` ensures that no objects are found by the statement sequence given in parentheses. Therefore "no objects" is the new value for the association.

```
…
CLASS Room
BIND Room_To_Component (VIA Object_To_Children CLASS Component)
```

Adds all components that are contained in a room to the room contents (given by all components associated with `Room_To_Component`). `ADD` has been omitted, because it is the default value.

1.6.12 EXECUTEACTION

Syntax: `'EXECUTEACTION' StringValue.`

Executes a named action for each object in the input set. Named actions can by defined in the model explorer.

Note: the objects in the input set are iterated **backwards**. If you want to iterate the input set in the predefined sequence, use `LOOP FOREACH (EXECUTEACTION ).`

Examples

```
…
CLASS demo_MoveOrder
EXECUTEACTION "DoMove"
```

Executes the action "DoMove" for each move order.

```
…
CLASS demo_MoveOrder
EXECUTEACTION "=ATTRIBUTE('demo_NextActionName')"
```

Executes for each move order the action that is specified by the text attribute `demo_NextActionName`.

1.6.13 EXECUTENAV >= v4.11.2

Syntax: `'EXECUTENAV' ['MODIFY'] StringValue.`

Builds and executes the FilterNavigation given by *StringValue*, which can be a string, a parameter or an expression.

If the executed FilterNavigation modifies the database 'MODIFY' must be set.

If *StringValue* is an expression and if this expression contains "ATTRIBUTE", the called FilterNavigation is built and executed for each object of the input set separately. In all other cases the called FilterNavigation is built and executed for all objects of the input set at once.

Examples

```
…
CLASS demo_MoveOrder
EXECUTENAV MODIFY DATAOF "ExecuteMoveOrder"
```

Executes the FilterNavigation in the Parameter "ExecuteMoveOrder".

```
…
CLASS demo_MoveOrder
EXECUTENAV "=ATTRIBUTE('bisU_MovedObjectsNavigation')"
```

Executes for each move order the FilterNavigation that is specified by the text attribute `bisU_MovedObjectsNavigation`.

Handling parameters from "outside": an "inside" FilterNavigation can only access parameters from "outside" which were stored in the global storage (SAVEAS GLOBAL). Careful: as documented in SAVEAS-Section, the standard tree search does not provide global storage.

1.7 Model Modification >= v5.2.2

All model modification **must be used very carefully**. They may **only** be used within a BMO file or within the search form if the enabler "MODEL" is present.

1.7.1 DELETE

Syntax: `'MODEL' 'DELETE' StringValue [StringValue].`

Deletes a class, an attribute, an association or a synonym or removes a property (second parameter) from a class.

If a model object with this identifier is not found, a warning is written to the log file.

If two parameters are given, the first parameter must be a class and the second parameter must be a property (attribute or association). In this case the property is removed from the class.

Example:

```
MODEL DELETE "Person" "bisB_ZipCode"
MODEL DELETE "bisB_Barcode"
```

1.7.2 CREATESYNONYM

Syntax: `'MODEL' 'CREATESYNONYM' StringValue StringValue.`

Creates a synonym with the first parameter as old name and the second parameter as new name.

Example:

```
MODEL CREATESYNONYM "bisB_Flooring" "bisC_Flooring"
```

1.7.3 RENAME

Syntax: `'MODEL' 'RENAME' StringValue StringValue.`

Renames a class, an attribute or an association. A corresponding synonym is created as in the model explorer. If a model object with this identifier is not found, a warning is written to the log file.

Example:

```
MODEL RENAME "ORG_Nachname" "bisB_LastName"
```

1.7.4 DELETEMETHOD

Syntax: `'MODEL' 'DELETEMETHOD' StringValue StringValue ['SET'].`

Deletes a method in the model. The first parameter identifies a class. The second parameter either identifies an attribute, an association, a named method or has one of the following values:

- "ONCREATE"
- "ONCOPY"
- "ONDESTROY"
- "ONMODIFY"

If the setter method of an attribute or association is to be deleted, the directive `'SET'` must be appended.

If the method cannot be found (class not found, attribute not found etc.), a warning is written to the log file.

Examples:

```
MODEL DELETEMETHOD "Person" "bisB_PersonPhoto" – Attribute Getter
MODEL DELETEMETHOD "Person" "bisKL_KeyHolderToKeyOp" SET – Assoc Setter
MODEL DELETEMETHOD "bisP_Order" "OrderPrepare" – class method
MODEL DELETEMETHOD "bisP_Order" "ONCREATE" – class method
```

1.7.5 SETDEFAULTVALUE

Syntax: `'MODEL' 'SETDEFAULTVALUE' StringValue StringValue [Value].`

Sets the default value of an attribute to a specific value. If Value is not present, the default value is removed.

Examples:

```
MODEL SETDEFAULTVALUE "Person" "Object_Display" "=NULL" – no value
MODEL SETDEFAULTVALUE "Person" "Object_Display" – no value
MODEL SETDEFAULTVALUE "bisP_Order" "bisP_Status" 0
```

2 Examples / Best Practice

2.1 Delete all objects of a specific class in the root container

```
-- delete all profm_Inters object under the root object
LOOP (
  BLOCK
  SAVEAS "ToDelete"
  ITERATE "profm_Inters" (
    [REFERENCES "Object_To_Parent" (BLOCK) OR RECALL "ToDelete"]
    SAVEAS "ToDelete"
    IF "COUNT > 1000" (BREAK)
  )
  RECALL "ToDelete"
  DELETE
) 10 -- 10 x 1000 = 10’000
```

2.2 All rep_Auftrag objects within a container

```
VIA "Object_To_Children" CLASS "rep_Auftrag"
```

2.3 All rep_Auftrag objects within a container or any subcontainers:

```
LOOP (VIA "Object_To_Children") CLASS "rep_Auftrag"
```

2.4 All Rooms and Buildings:

```
START INSTANCES "Room"
[ START INSTANCES "Building" OR CLASS "Space" ]
```

2.5 All orders in descending order of there type, same types ascending deadlines

```
START INSTANCES "bisM_Order"
SORT ( -"Object_Type"  +"bisO_Deadline" )
```

2.6 XDS: All bisM_Order objects Deadline < the value of Variable $vartime$.

```
START VALUE "bisO_Deadline" &lt; DATAOF "vartime"
CLASS "bisM_Order"
```

There is a variable definition for vartime for example:

```
<Variables>
  <Var Name="vartime"  Value="=NOW + 200"/>
</Variables>
```

2.7 BISWeb – Searching a Person via indexed attribute:

```
START VALUE "ORG_Nachname" ~= "=searchKey + '*'"
CLASS "ORG_Personal_Einheit"
```

Note: "searchKey" must be a parameter given by BisWeb2.

2.8 BISWeb – Searching a room or a workplace (subroom) via indexed attribute:

```
START VALUE "bisB_SpaceId" "=searchKey + '*'"
CALSS "Room"
[ VIA "Object_To_Children" OR CLASS "Room"]
```

Note: "searchKey" must be a parameter given by BisWeb.
The union (`[..]`) must be used in order not to loose the rooms found by the `start` statement.

2.9 BISWeb – Navigating to all rooms (I):

```
START CLASS "Space"
LOOP (VIA "Object_To_Children")
CLASS Room
```

Note: The following statement is implicit added through BISWeb and must only be used in the formula if it's different as follows:
`VALUE "Object_Display_Kontext" = "=searchKey + '*'"`

"searchKey" must be a parameter given by BisWeb.

Searching with an index (`START VALUE`) is more performant.

2.10 BISWeb – Navigating to all rooms (II):

With this statement there is actually no navigation done, but all Rooms are accessed at once.

```
START INSTANCES "Room"
```

Note: The following statement is implicit added through BISWeb and must only be used in the formula if it's different as follows:
`VALUE "Object_Display_Kontext" = "=searchKey + '*'"`

"searchKey" must be a parameter given by BisWeb.

Searching with an index (`START VALUE`) is more performant.

2.11 Select Orders with a Date Range for a responsible Person

Consider you want to show orders in a planner or a gantt view. For each person in a maintenance group his or her orders are shown in a date range given by the parameters *VAL_FirstDate* and *VAL_LastDate*.

Byron/BIS >= v5.0.3 – Everything can be executed with the `START VALUE` statement using the index on *bisT_StartDateTime*. If the association bisP_MaintToMaintainer is indexed as well, this indexed will be used too and the `ANDCLASS` might not be necessary.

```
SAVEAS "Person"
START VALUE "bisT_StartDateTime" >= "=VAL_FirstDate -7"<= "=VAL_LastDate +1"
ANDCLASS "bisP_Order"
ANDREFERENCES "bisP_MaintToMaintainer" (RECALL "Person")
```

Byron/BIS < 5.0.3 - using the index on *bisT_StartDateTime*, all orders in the requested date range can be retrieved quickly. Orders assigned to other persons can be removed with the `COUNT` statement.

```
SAVEAS "Person"
START VALUE "bisT_StartDateTime" >= "=VAL_FirstDate -7"<= "=VAL_LastDate +1"
CLASS "bisP_Order"
COUNT ([VIA "bisP_MaintToMaintainer" AND RECALL "Person"]) <> 0
```

3 VALUE Statement Notes

To compare the attribute with a variable value, use the 'DATAOF' clause.

For each data type, a complete list of examples is given. The compare operations are:

Sign	Operation
NOT	Not match (with pattern)
=	Equals
<	Smaller
>	Greater
<=	Smaller or equal
>=	Greater or equal
<>	Not equal
~=	Match (with pattern)
!~	Not match (with pattern)
^=	Exists (a value must be given)
!^	Not exists (a value must be given)

NB: The not equal operator ("<>") and the not match operator ("!~") is true for not existing values too!.

If the text is embedded within a xml definition some of this signs must be written in the & notation. For example instead of **<** use **`<`**

3.1 Text

Operation	Example
NOT / !~	VALUE bisB_Description NOT "*ActiveX*"
=	VALUE bisB_Description = "Hallo"
<	VALUE Name < "B"
>	VALUE Name > "B"
<=	VALUE Name <= "B"
>=	VALUE Name >= "B"
<>	VALUE bisB_Description <> "Hallo"
~=	VALUE bisB_Description ~= "*ActiveX*"
^=	VALUE bisB_Description ^= xxx
!^	VALUE bisB_Description !^ xxx

3.2 Numbers (Integer / Floating Point)

Operation	Example
NOT / !~	n.a.
=	VALUE bisW_Index = 0
<	VALUE bisW_Index < 5
>	VALUE bisW_Index > 5
<=	VALUE bisW_Index <= 5
>=	VALUE bisW_Index >= 5
<>	VALUE bisW_Index <> 0
~=	n.a.
^=	VALUE bisW_Index ^= xxx
!^	VALUE bisW_Index !^ xxx

3.3 Enumeration

Instead of textual values, numerical values may be used as well.

Operation	Example
NOT / !~	n.a.
=	VALUE rep_AuftragsStatus = "neu erfasst" VALUE rep_AuftragsStatus = 0
<	VALUE rep_AuftragsStatus < "in Arbeit" VALUE rep_AuftragsStatus < 2
>	VALUE rep_AuftragsStatus > "in Arbeit" VALUE rep_AuftragsStatus > 2
<=	VALUE rep_AuftragsStatus <= "in Arbeit" VALUE rep_AuftragsStatus <= 2
>=	VALUE rep_AuftragsStatus >= "in Arbeit" VALUE rep_AuftragsStatus > 2
<>	VALUE rep_AuftragsStatus <> "erledigt" VALUE rep_AuftragsStatus <> 5
~=	n.a.
^=	VALUE rep_AuftragsStatus ^= xxx
!^	VALUE rep_AuftragsStatus !^ xxx

3.4 Dates

Operation	Example
NOT / !~	n.a.
=	VALUE rep_Erfasst = "15.11.2001"
<	VALUE rep_Erfasst < "15.11.2001"
>	VALUE rep_Erfasst > "15.11.2001"
<=	VALUE rep_Erfasst <= "15.11.2001"
>=	VALUE rep_Erfasst >= "15.11.2001"
<>	VALUE rep_Erfasst <> "15.11.2001"
~=	n.a.
^=	VALUE rep_Beendet ^= xxx
!^	VALUE rep_Beendet !^ xxx

Dates must be entered according to your regional settings.

Alternatively dates can be expressed as numbers (e.g. 37210.0 for 15.11.2001 – You can use EXCEL to obtain the numbers (first set the cell format to „Date", enter a date then set the format to number. The fraction of the number gives the time - .0 for midnight, .5 for noon).

3.5 Interval

Operation	Example
NOT / !~	n.a.
=	VALUE rep_Duration = "2:00"
<	VALUE rep_Duration > "2:00"
>	VALUE rep_Duration < "2:00"
<=	VALUE rep_Duration <= "2:00"
>=	VALUE rep_Duration >= "2:00"
<>	VALUE rep_Duration <> "2:00"
~=	n.a.
^=	VALUE rep_Duration ^= xxx
!^	VALUE rep_Duration !^ xxx

Alternatively intervals can be expressed as numbers (e.g. 1.0 for one day; 0.5 for 12 hours, …).

3.6 Boolean (logical value)

Operation	Example
NOT / !~	n.a.
=	VALUE rep_Oeffentlich = true

	VALUE rep_Oeffentlich = false
<	n.a.
>	n.a.
<=	n.a.
>=	n.a.
<>	VALUE rep_Oeffentlich <> true VALUE rep_Oeffentlich <> false
~=	n.a.
^=	VALUE rep_Oeffentlich ^= xxx
!^	VALUE rep_Oeffentlich !^ xxx

Alternatively Booleans can be expressed as numbers (0 for false, anything else for true).

4 Pseudo Parameter

Pseudo Parameter are read-only global parameter. The value is not really fetched in the global parameter, but it is evaluated. This is the reason you cannot see this parameter in the definition list. There are the following parameters:

Name	Description	Example Value
BIS_BuildVersion	Full Version Number of exe	4.7.0.490
BIS_ReleaseVersion	Version Number of exe	4.7.0
BIS_DbVersion	Database Version	20100301
BIS_DbAlias	Database Alias **(>= v4.9.8)**	Demo
BIS_DbPath	Database Path **(>= v4.9.8)**	C:\Dokumente und Einstellungen\All Users\ByronBIS\DemoDb\Demo.bisdb

This Parameter can be used like normal parameters:

```
MODIFY Name "=PARAM('BIS_DbVersion')"
```