Version 1.2
June 15, 2015

Matrix42
ASQL Reference

Copyright

Content

1 Introduction

This document provides an introduction to the Matrix42 query language ASQL and covers the most important functions in everyday use.
Within the Matrix42 Service Store ASQL is used for defining:

- Column Definitions of Object Searches
- Calculated fields
- Display expressions
- Filter criteria
- Structures
- Quick Filter
- Script Definition Parameter Expressions
- Compliance Rules Conditions

We use ASQL expression in two different contexts. First, to define the attributes we want to query from a class (the base class) and its related classes. Second to restrict the result of a query to a subset of the instances (rows) of the base class. We call the first a display expression and the second a WHERE clause. These two share in principle the same syntax with four differences:

- A display expression contains, separated with commas, often more than one expression, a where clause always only contains one expression.
- The expressions in a display expression must define an alias if they are not direct attributes of the base class. This alias is used to name the columns in the result and must be unique in the result.
- The expressions in a display expression must have unique results for each instance (row) of the base class. You cannot display the children of a 1-N relation, because there may be more than one child.
- A WHERE clause must always be of type Boolean expression

2 Simple Queries for display expressions

An ASQL display expression serves to define the attributes we want to query from a class (the base class) and its related classes. It is build up expressions containing:

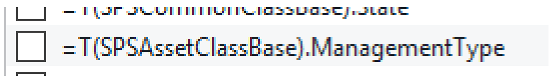
- Identifiers of data definitions (classes) or attributes (including relation attributes)
- Functions
- Constants
- Operators

Example:
The simplest query for display expressions in ASQL is the query of attributes of an object type class base.
The object type (configuration item) “Computer“ (`SPSComputerType`) has the class base `SPSComputerClassBase`.
Valid ASQL queries for e.g. being used for column definitions could be:

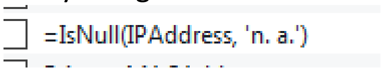
- `ManagementType` Value of computer attribute ManagementType
- `‘KA_’+Name` Value of computer attribute Name leaded by the string ‘KA’
- `IsNull(IPAddress, ‘IP not available’)` Value of computer attribute IPAddress if not Null, else the string ‘IP not available’

Tip: If you are already familiar with normal SQL, in most cases you just can imagine to place these ASQL queries between “`SELECT`” and “`from SPSComputerClassBase`” and you will get the according SQL query.
At some places ASQL display expression can be used, e.g. the column definitions, the expression must be leaded with a “=”.

“=” is needed for attributes stored in Relations


=T(SPSAssetClassBase).ManagementType

Or by using a function:


=IsNull(IPAddress, 'n. a.')

3 Simple Queries for WHERE clauses

An ASQL WHERE clause serves to restrict the result of a query to a subset of the instances (rows) of the according base class. It is build up expressions containing:

- Identifiers of data definitions (classes) or attributes (including relation attributes)
- Functions
- Constants
- Operators

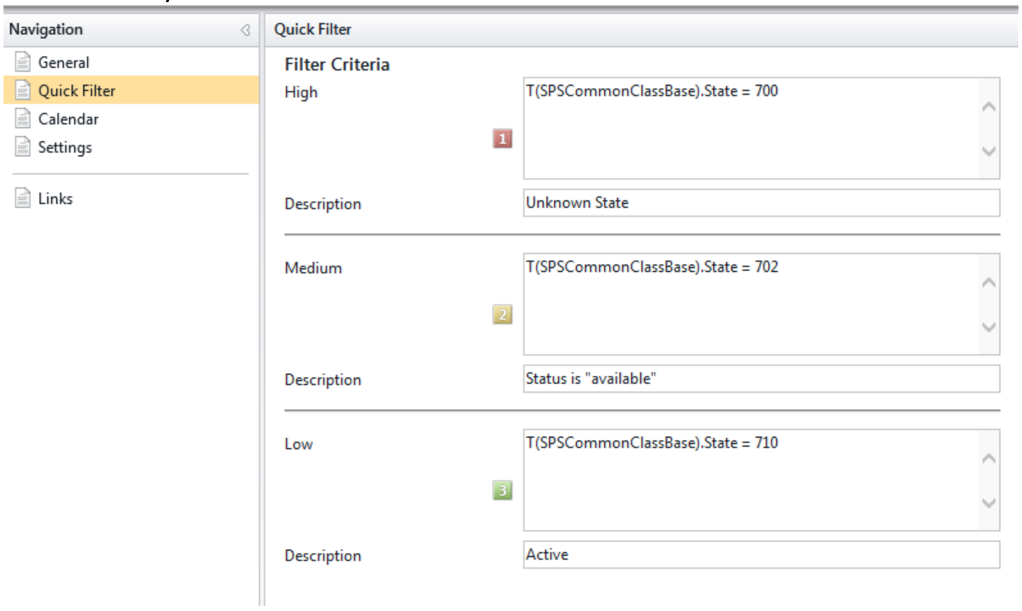
Example:
The simplest query for WHERE clauses in ASQL is the query of attributes of an object type class base. The object type (configuration item) "Computer" (`SPSComputerType`) has the class base `SPSComputerClassBase`.
Valid ASQL queries for e.g. being used for quick filters could be:

- `ManagementType=0` All Computers where ManagementType = 0 [manually]
- `Name like 'KA%'` All Computers where the name starts with KA
- `IPAddress IS NOT NULL` All computers that have an IP address

Tip: If you are already familiar with normal SQL, in most cases you just can imagine to add "`SELECT * from SPSComputerClassBase WHERE`" at the beginning of these ASQL queries and you will get the according SQL query.

It can be used in static/dynamic structures, Quickfilters, Mailrecipient, (CoRu/WF-Designer), et cetera.

e.g.:
QuickFilter by state



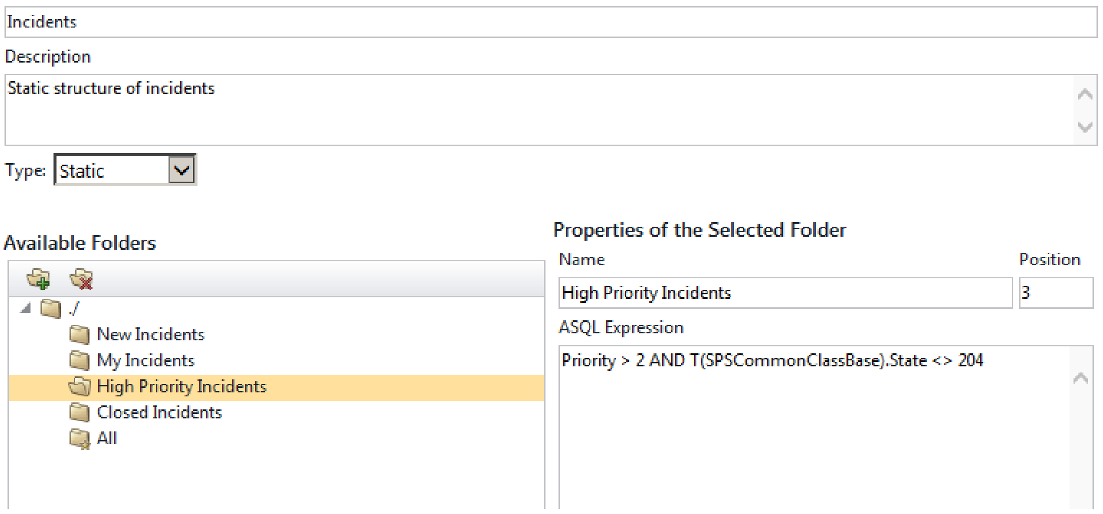
Filter Criteria

High: T(SPSCommonClassBase).State = 700
Description: Unknown State

Medium: T(SPSCommonClassBase).State = 702
Description: Status is "available"

Low: T(SPSCommonClassBase).State = 710
Description: Active

Static structure:



Incidents

Description

Static structure of incidents

Type: Static

Available Folders

- ./
  - New Incidents
  - My Incidents
  - High Priority Incidents
  - Closed Incidents
  - All

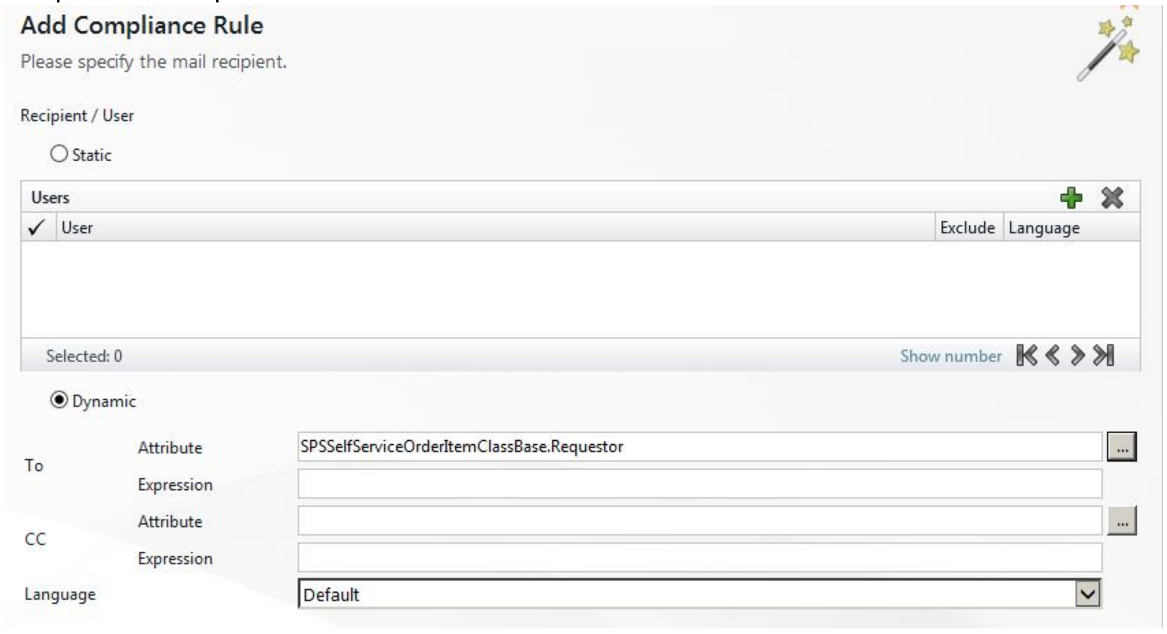
Properties of the Selected Folder

Name	Position
High Priority Incidents	3

ASQL Expression

```
Priority > 2 AND T(SPSCommonClassBase).State <> 204
```

Recipient in Compliance Rule



Add Compliance Rule

Please specify the mail recipient.

Recipient / User

○ Static

Users

✓	User	Exclude	Language

Selected: 0 Show number

● Dynamic

To	Attribute	SPSSelfServiceOrderItemClassBase.Requestor
	Expression	
CC	Attribute	
	Expression	
Language		Default

4 Queries across several Class Bases (Navigate relations)

By default ASQL queries are always used in a defined context of an object type (configuration item) and one specific class base (data definition).
The simplest relation navigation mechanism is to access related classes defined in the data model as relations from the base class is to use the relation name. So for navigating relations starting from the base class you can use the simple "."-syntax:
<RelationName>.<AttributeName>
Examples (context SPSComputerClassBase):

- `AssignedAccounts.AccountName like 'XYZ'` All computers which are assigned to the account XYZ
- `AssignedAccounts.Owner.ID IS NOT NULL` All computers which have a person assigned

But additionally an object type (configuration item) may have several class bases (data definitions) with attributes.
Example:
The object type `SPSComputerType` contains the following class bases:

- `SPSComputerClassBase` Class base tor computers
- `SPSAssetClassBase` Class base tor assets
- `SPSlnventoryClassBase` Class base for inventory data

To access classes that are declared in the according object type but are not the base class in context the T-Operator is used (see the following chapter for details).
In some special cases, like e.g. on defining Compliance Rule conditions, there is no specific class base in context, but just an object type. In these cases, you can navigate to all contained classes in the type in context using the expression:
Related<ClassName>.<AttributeName>
Examples (context SPSComputerType):

- `RelatedSPSAssetClassBase.lnventoryNumber` Inventory Number of the computer
- `RelatedCommonClassBase.Location.Name` Location of the computer
- `RelatedComputerClassBase.ManagementType=0` All computers where ManagementType = 0

4.1 T-Operator

You can reach attributes as well as relations with the T-Operator.
Syntax:
T(<ClassName>).<AttributeName>
T(<ClassName>).<RelationName>.<AttributeName>
Examples:

- `T(SPSAssetClassBase).lnventoryNumber` Inventory number of a computer
- `T(SPSCommonClassBase).Location.Name` Location name of a computer

As relations only contain the ID of the other target object, you can continue from there to the desired target attribute.
In the above examples, `lnventoryNumber` is only an attribute of the asset class base and the assigned `Location` is the name of the relation from the common class base to the location class base.
Relations only store the ID. In the case of the principal user relation, the asset class base contains the ID of the user class base.
The two classes [asset & user] are fully connected and by using the relation name, all attributes of the related class can be reached.
You can either simply define the target attribute you want to display, or modify the display name by an expression.

Example:

Display of the first name of the main user:

```
T(SPSAssetClassBase).AssignedUser.FirstName
```

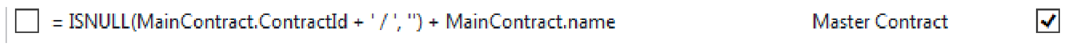
Create a display expression:

```
T(SPSAssetClassBase).AssignedUser[LastName + ISNULL(‘, ’+ FirstName,’’)]
```

If you do not define a target attribute, the system will show by default the display expression defined in the target class.

Example 2:

It´s also possible to display attributes across Datadefinitions with these queries in Display-Layouts.



☐	= ISNULL(MainContract.ContractId + ' / ', '') + MainContract.name	Master Contract	✓

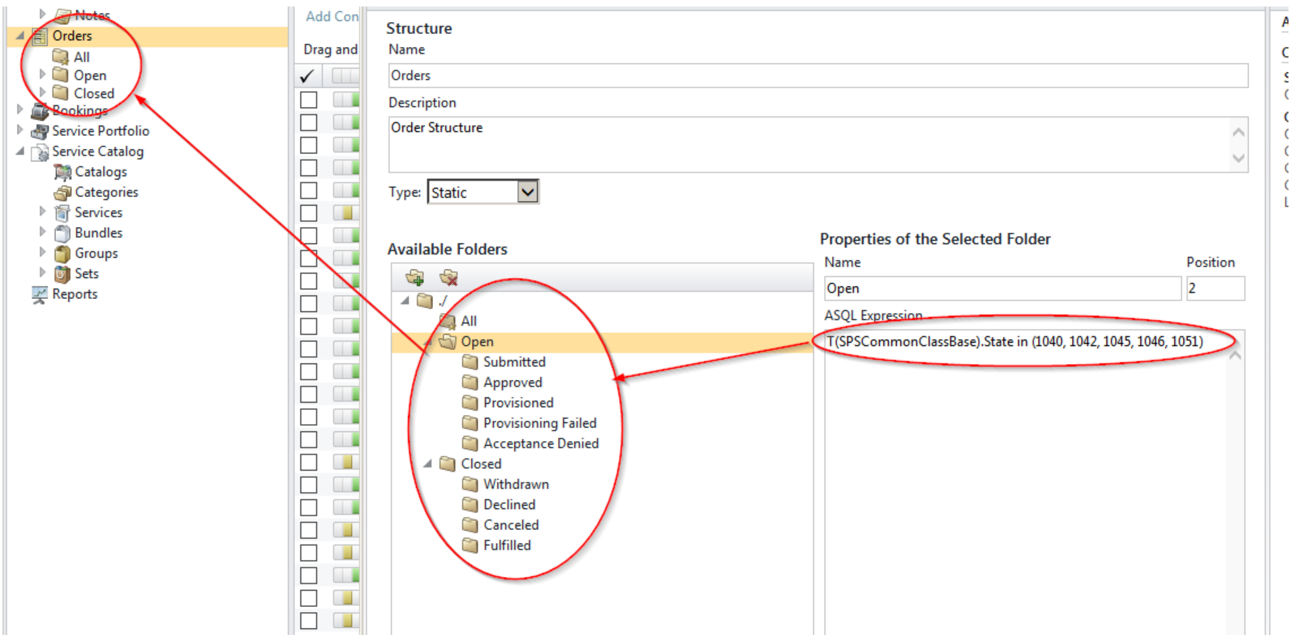
5 Structures

Structures can use ASQL-Queries for Pre-filtering Data.

5.1 Static Structures

Static Structures can used very similar to the Quick Filters.

For Example you can simply filter by state



Structure

Name: Orders

Description: Order Structure

Type: Static

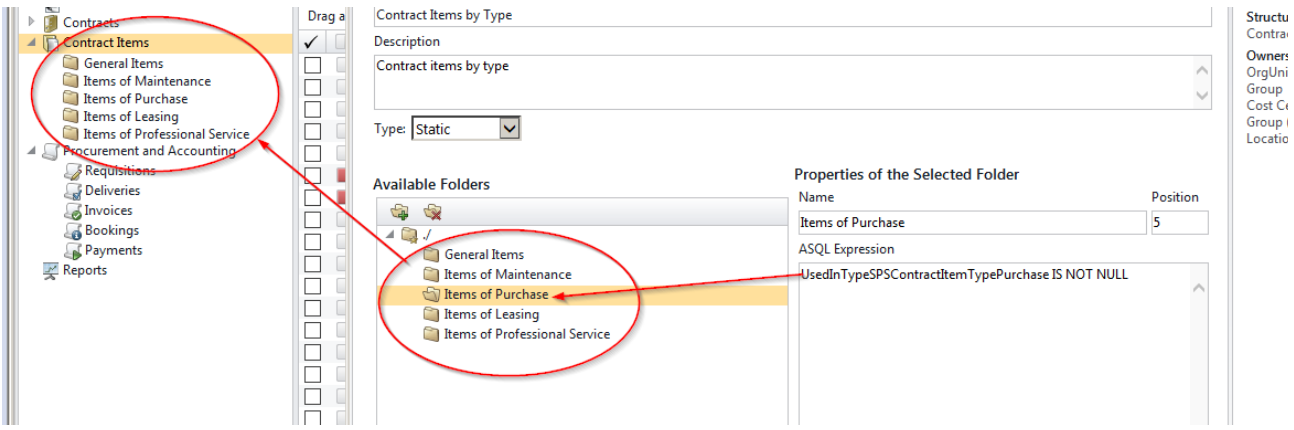
Properties of the Selected Folder

Name: Open

Position: 2

ASQL Expression: T(SPSCommonClassBase).State in (1040, 1042, 1045, 1046, 1051)

or UsedIntype.



Contract Items by Type

Description: Contract items by type

Type: Static

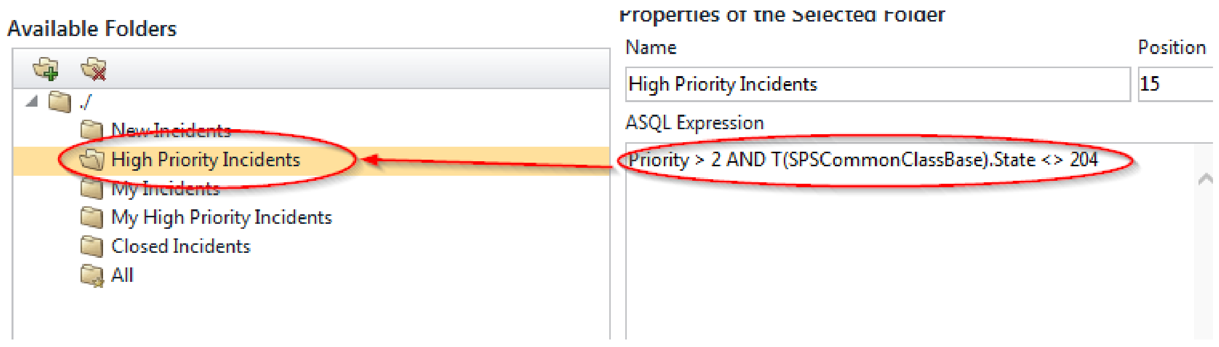
Properties of the Selected Folder

Name: Items of Purchase

Position: 5

ASQL Expression: UsedInTypeSPSContractItemTypePurchase IS NOT NULL

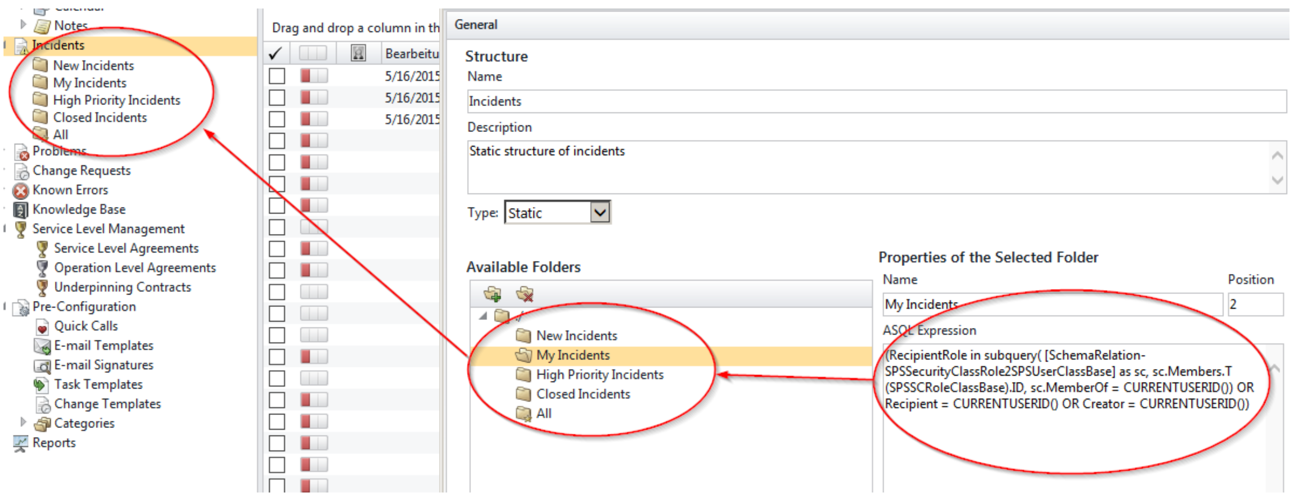
Combination of State and Pickup-Values,



Name: High Priority Incidents
Position: 15
ASQL Expression:

```
Priority > 2 AND T(SPSCommonClassBase).State <> 204
```

more complex queries

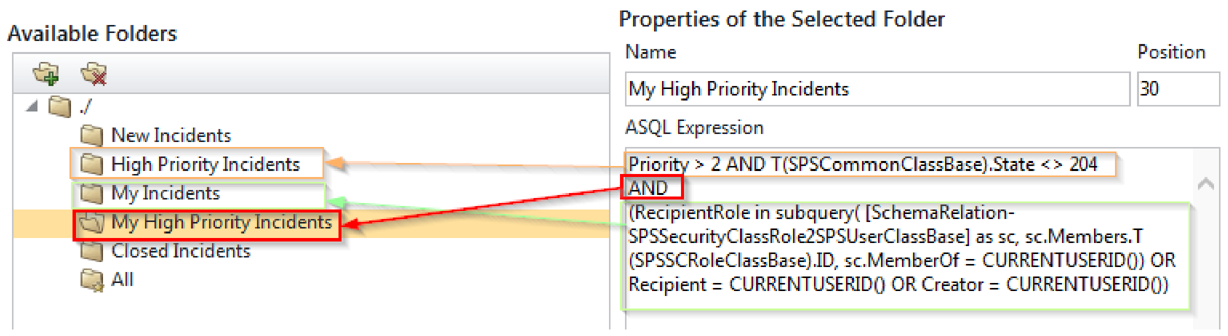


Structure
Name: Incidents
Description: Static structure of incidents
Type: Static

Name: My Incidents
Position: 2
ASQL Expression:

```
(RecipientRole in subquery( [SchemaRelation-
SPSSecurityClassRole2SPSUserClassBase] as sc, sc.Members.T
(SPSSCRoleClassBase).ID, sc.MemberOf = CURRENTUSERID()) OR
Recipient = CURRENTUSERID() OR Creator = CURRENTUSERID())
```

And a combination of these.



Name: My High Priority Incidents
Position: 30
ASQL Expression:

```
Priority > 2 AND T(SPSCommonClassBase).State <> 204
AND
(RecipientRole in subquery( [SchemaRelation-
SPSSecurityClassRole2SPSUserClassBase] as sc, sc.Members.T
(SPSSCRoleClassBase).ID, sc.MemberOf = CURRENTUSERID()) OR
Recipient = CURRENTUSERID() OR Creator = CURRENTUSERID())
```

5.2 Dynamic structures

Naming:

By default the structure can use attributes from specified class base (data definition) for naming structure-folders. It is not allowed to use relation to other Cis for naming folders.

You can build flat or hieratic structures.

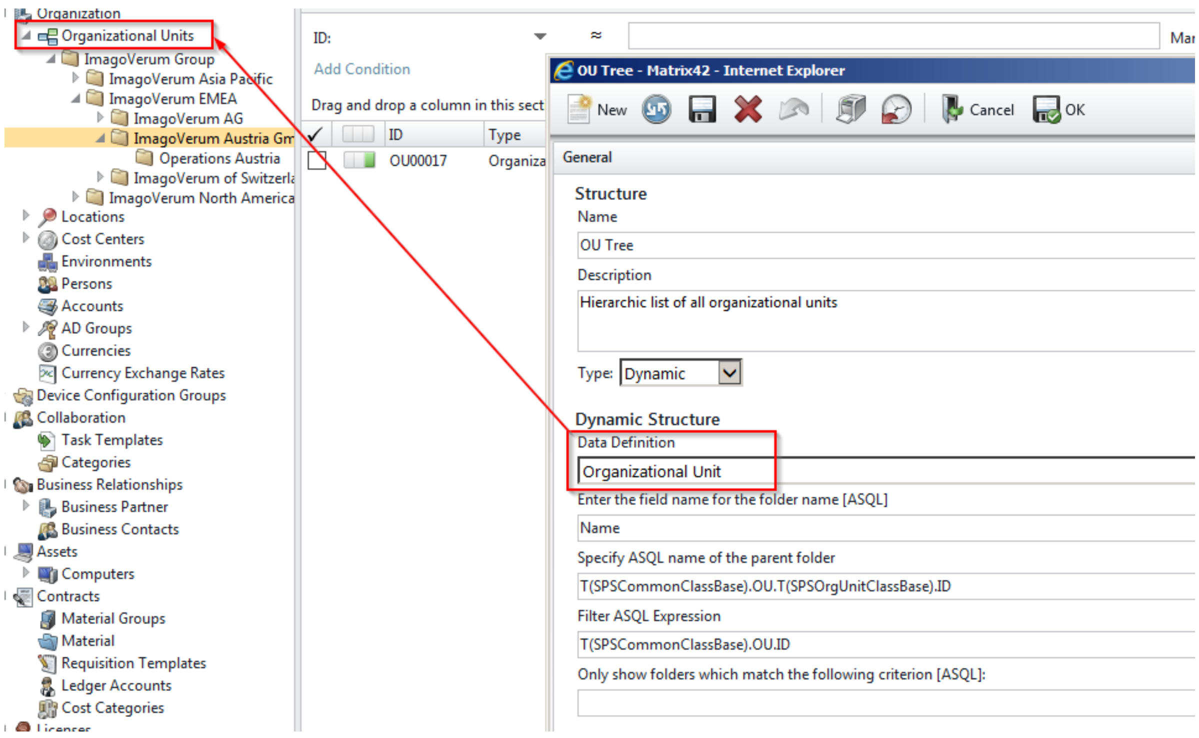
ASQL - Dokumentation

The names of the structure-folders have to be in the choosen data definition.

Parent folder can be defined in other DDs.

ASQL Filter Expression have to be the ID of the attributes used for the structure like the attribute ID of the choosen data definition containing the Parent relation.

e.a. SPSCommonClassBase.OU



Structure

Name: OU Tree

Description: Hierarchic list of all organizational units

Type: Dynamic

Dynamic Structure

Data Definition: Organizational Unit

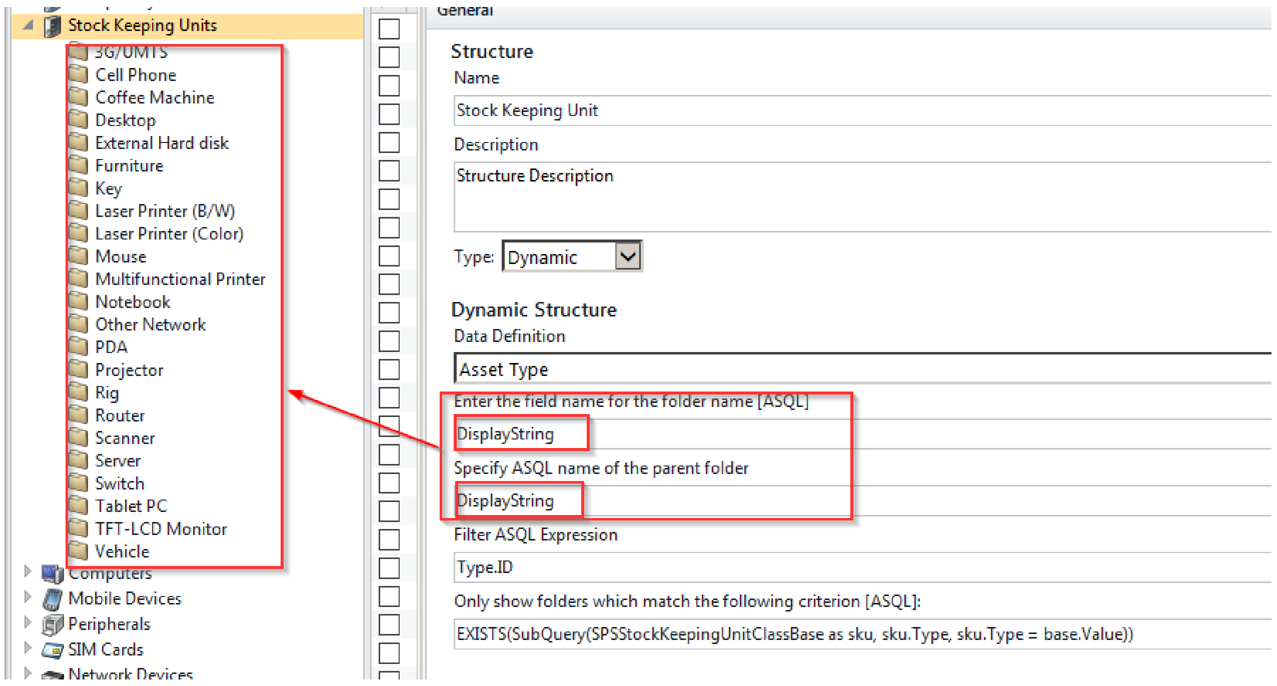
Enter the field name for the folder name [ASQL]: Name

Specify ASQL name of the parent folder: T(SPSCommonClassBase).OU.T(SPSOrgUnitClassBase).ID

Filter ASQL Expression: T(SPSCommonClassBase).OU.ID

Only show folders which match the following criterion [ASQL]:

Flat structures did not have a Parent relation



Structure

Name: Stock Keeping Unit

Description: Structure Description

Type: Dynamic

Dynamic Structure

Data Definition: Asset Type

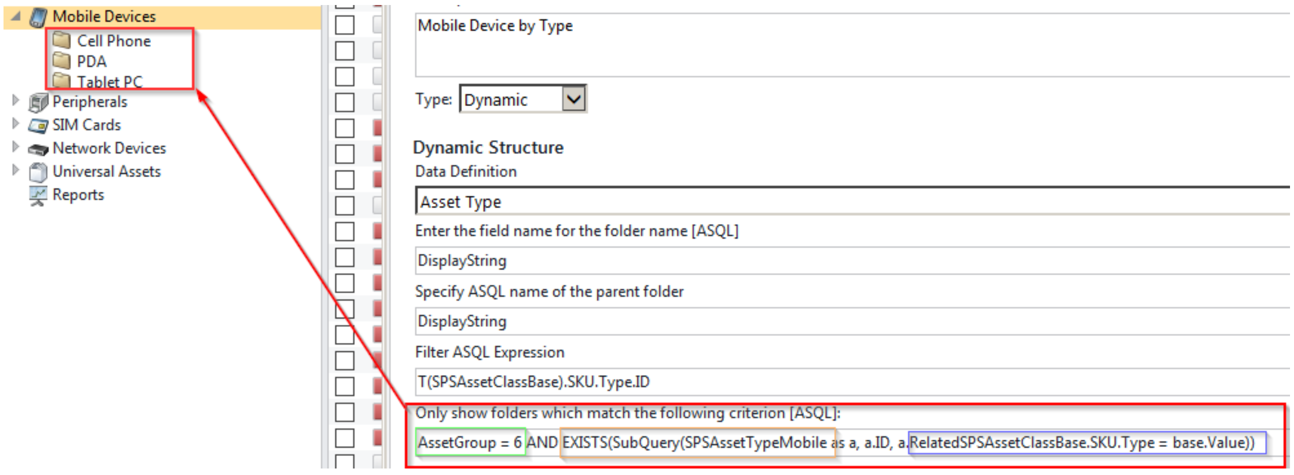
Enter the field name for the folder name [ASQL]: DisplayString

Specify ASQL name of the parent folder: DisplayString

Filter ASQL Expression: Type.ID

Only show folders which match the following criterion [ASQL]: EXISTS(SubQuery(SPSStockKeepingUnitClassBase as sku, sku.Type, sku.Type = base.Value))

Additionally you can use a where-clause with ASQL-queries.



Mobile Device by Type

Type: Dynamic

Dynamic Structure

Data Definition: Asset Type

Enter the field name for the folder name [ASQL]: DisplayString

Specify ASQL name of the parent folder: DisplayString

Filter ASQL Expression: T(SPSAssetClassBase).SKU.Type.ID

Only show folders which match the following criterion [ASQL]:

```
AssetGroup = 6 AND EXISTS(SubQuery(SPSAssetTypeMobile as a, a.ID, a.RelatedSPSAssetClassBase.SKU.Type = base.Value))
```

6 Operators

- Arithmetic operators
  - **+, -,*,/**
- Binary operators
  - **|, &, ^**
- Compare operators
  - **=, <=, =<, =>, >=, <, >, <>, LIKE, IS [NOT] NULL**
- Logical operators
  - **AND, OR, NOT**
- Subquery operators
  - **IN, EXISTS**

7 Functions

7.1 The SUBQUERY function

The SUBQUERY function is used to query related classes or to limit the result set.
Syntax:
SUBQUERY (<BaseClass> AS <Alias>, <TargetAttribute>, <Filter>)
In which:

- **<BaseClass>** BaseClass for the SubQuery
- **<Alias>** The alias will be used for the <TargetAttribute> and <Filter>
- **<TargetAttribute>** Defines the result set for the Subquery
- **<Filter>** Limits the result set

Example:
We want a column in the list view for persons (`SPSUserType` with `SPSUserClassBase`) to show the number of assets they have assigned.

```
=SUBQUERY(SPSAssetClassBase AS ACB, Count(all, ACB.*), 
ACB.AssignedUser.ID=base.ID)
```

7.2 The CASE function

The CASE function allows you to discriminate between various cases depending on the values displayed.
Syntax:
CASE WHEN <attribute> = <value> THEN ‘Condition1 met‘ ... ELSE ‘No condition met‘ END
In our example we will use the SQL function DateDiff(), which calculates the time difference between an attribute and the current date (getdate()) in the context of `SPSContractItemClassBase`.

```
=CASE WHEN DateDiff(month, getdate(), ValidUntil) < 0 THEN ‘expired’ ELSE 
‘active’ END
```

7.3 Special functions

- **CurrentUserID() CurrentUserID ( )** Returns the ID column of the SPSUserBaseClass for the current logged in user in Console.

- **InteractiveUserID() InteractiveUserID ( )** Returns the ID column of the SPSUserBaseClass for the current logged in user in portal.

- **Recursive() RECURSIVE (relation_identifier)** Provides all object instances of an hierarchical tree structures that are part of a parent or child branch. Can be used only for the following classes (data definitions): SPSOrgUnitClassBase, SPSLocationClassBase, SPSScCategoryClassBase, SPSArticleCategoryClassBase. Examples:
o `Recursive(OUFor).T(SPSOrgUnitClassBase).Name='XYZ'` Context: SPSOrgUnitClassBase. Provides all Organization Units that are part of the parent branch of the Organization Unit XYZ.
o `T(SPSCommonClassBase).Recursive(OU).T(SPSOrgUnitClassBase).Name='XYZ'` Context: SPSOrgUnitClassBase. Provides all Organization Units that are part of the child branch of the Organization Unit XYZ.
o `Recursive(Parent).Name='Service Desk'` Context: SPSScCategoryClassBase. Provides all Categories that are part of the parent branch of the Category Service Desk.

7.4 Standard SQL functions

ASQL supports the usage of all important functions provided natively by MS SQL Server's Transact-SQL syntax. Below you will find the list of supported functions. Please refer to http://msdn.microsoft.com/de-de/library/bb510741.aspx for detailed information about the single functions.

- **Count() COUNT ({ [[ALL | DISTINCT ,] expression] | * })** Returns the number of items in a group. COUNT always returns an int data type value. This function can be used only within the SUBQUERY function.
- **Sum() SUM (expression)** Returns the sum of all the values in the expression. SUM can be used with numeric columns only. Null values are ignored. This function can be used only within the SUBQUERY function.

- **IsNull() ISNULL (check_expression , replacement_value)** Replaces NULL with the specified replacement value.
- **Coalesce() COALESCE (expression [,...n])** Returns the first nonnull expression among its arguments.
- **Reverse() REVERSE (string_expression)** Returns the reverse of a string value.
- **Left() LEFT (character_expression , integer_expression)** Returns the left part of a character string with the specified number of characters.
- **Right() RIGHT (character_expression ,integer_expression)** Returns the right part of a character string with the specified number of characters.
- **LTrim() LTRIM (character_expression)** Returns a character expression after it removes leading blanks.
- **RTrim() RTRIM (character_expression)** Returns a character string after truncating all trailing blanks.
- **Len() LEN (string_expression)** Returns the number of characters of the specified string expression, excluding trailing blanks.
- **SubString() SUBSTRING (value_expression ,start_expression ,length_expression)** Returns part of a character, binary, text, or image expression.
- **Replace() REPLACE (string_expression,string_pattern,string_replacement)** Replaces all occurrences of a specified string value with another string value.
- **PatIndex() PATINDEX ('%pattern%' , expression)** Returns the starting position of the first occurrence of a pattern in a specified expression, or zeros if the pattern is not found, on all valid text and character data types.
- **GetDate() GETDATE ( )** Returns the current database system timestamp as a datetime value without the database time zone offset. This value is derived from the operating system of the computer on which the instance of SQL Server is running.
- **DateAdd() DATEADD (datepart , number , date)** Returns a specified date with the specified number interval (signed integer) added to a specified datepart of that date.
- **DateDiff() DATEDIFF (datepart ,startdate ,enddate)** Returns the count (signed integer) of the specified datepart boundaries crossed between the specified startdate and enddate.
- **DatePart() DATEPART (datepart , date)** Returns an integer that represents the specified datepart of the specified date.
- **NewID() NEWID ( )** Creates a unique value of type uniqueidentifier.
- **Cast() CAST (expression, data_type)** Explicitly converts an expression of one data type to another.
- **Min()** Min find the minimum Value in a equal to SQL Min-Function
- **Max()** Max find the minimum Value in a equal to SQL Max-Function

Amalienbadstraße 41 / Bau 54
76227 Karlsruhe
Germany
+49 (721) 914 32 - 300
Internet E-Mail: info@matrix42.de
Website: http://www.matrix42.de
Matrix42 Service Store, Version: 5.30