2. CONFIGURATION OF ADOXX COMPONENTS

QUERY

Platform basic functionalities in the analysis component

Predefined queries:

Professional queries, which are business or method specific defined. For the execution of these, no AQL knowledge required.

Relationtables:

Relationtables make relations (connectors or references) between two objects of same or different models.

[1] **AQL** = **A**DOxx **Q**uery **L**anguage

Predefined queries are…

- Queries on models, model content and dependencies of models
- Session-independent
- Defined by the user and therefore easy to use
- Specific for the used model and their modelling language
- A configuration of the basic functionalities of the ADOxx Platform. These queries can:
  - Be defined for models of a specific type
  - Combined into groups (topics)

Creation of method-specific Queries

Steps which are necessary:

1. Define menu item
2. Define input fields
3. Define AQL queries
4. Define result attributes

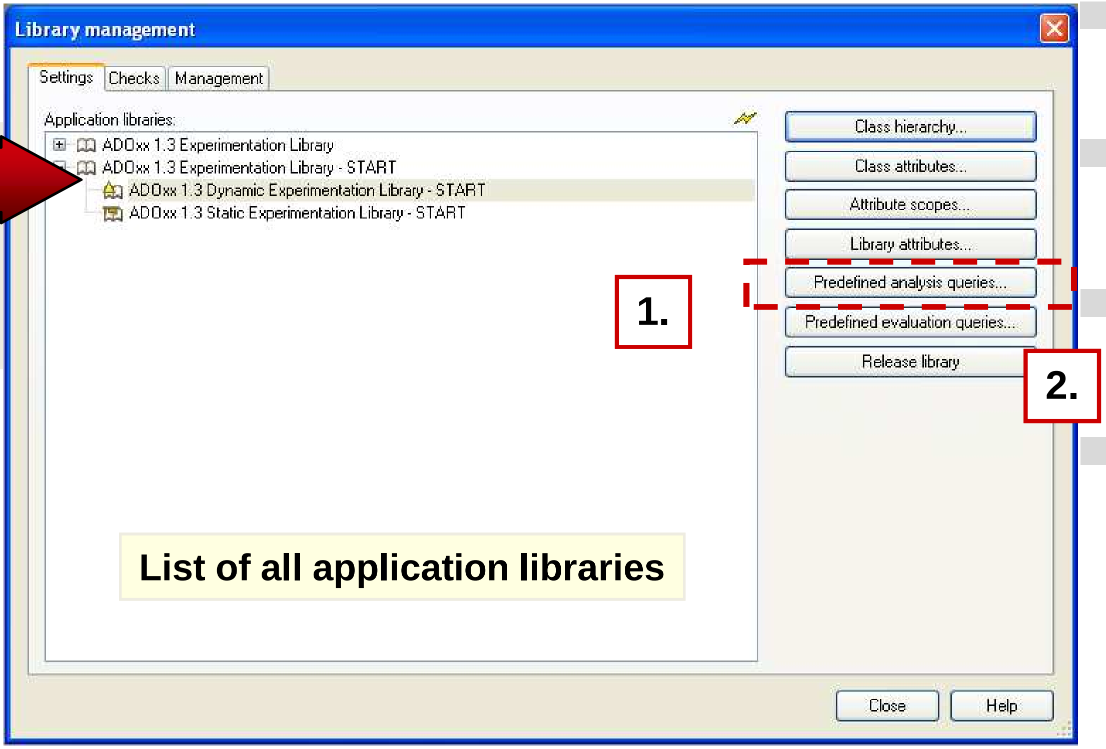
Start Analysis Query Function

Choose Library
(D- or S-Library)
„Predefined analysis queries“

- SmartIcon

or

- Menu **„Library“**
- Menu item **„Configuration“**



Library management

Settings | Checks | Management

Application libraries:

- ADOxx 1.3 Experimentation Library
- ADOxx 1.3 Experimentation Library - START
  - ADOxx 1.3 Dynamic Experimentation Library - START
  - ADOxx 1.3 Static Experimentation Library - START

List of all application libraries

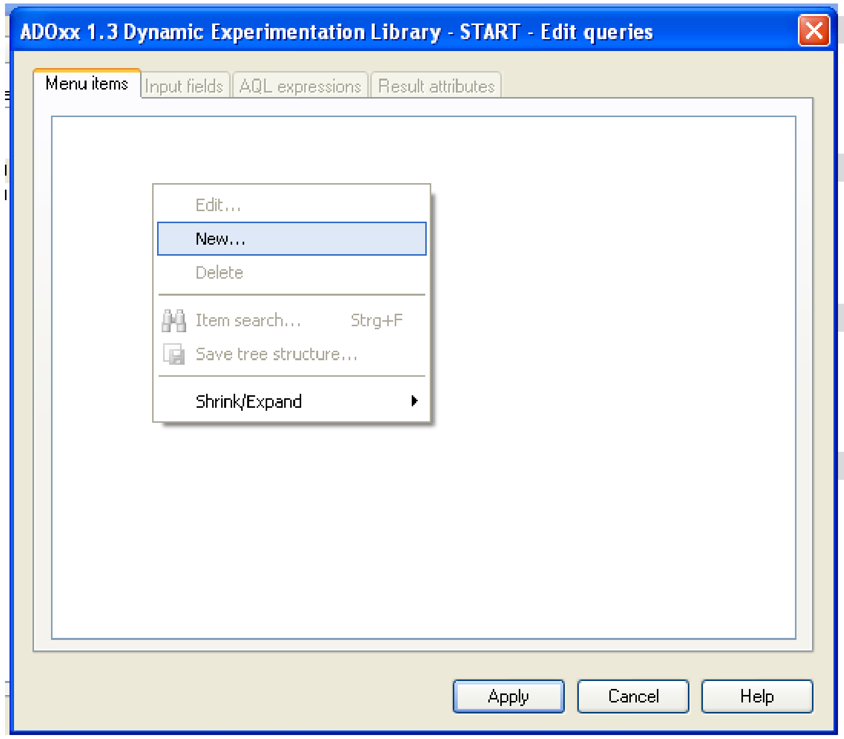
In order to edit the analysis queries you can use selection dialogs in the library management

Query Functions

Available functions:

- **New:** Create a new menu item or a new query
- **Edit:** Edit an existing query
- **Delete:** Delete an existing query from the library
- **Search entrys:** Call of the ADOxx search function
- **Save tree-structure:** Save the content of the text file
- **Show/close:** Diverse view options

The appeal of all these function is under the context menu of the list



AQL Notation

- Extended Backus Naur Form (EBNF) notation is used for describing the AQL syntax
- Terminal symbols
  - symbols which cannot be split up further
  - are included by inverted commas ´…´
- Non-terminal symbols
  - are included by <...>
- Symbols {...}, [...] and | serve to formulate rules in a more compact form :
  - **{...}** arbitrary number of iterations (even 0-times)
  - **[...]** optional (0- or 1-time)
  - **|** alternative
- Rules start with a non-terminal symbol, followed by "::=" and the symbol's definition

HINT: Keep in mind that AQL is cAse sEnsiTivE

AQL Terminal symbols (I)

‘<‘ and ‘>‘ in this order are used to represent a class, a relation, a model, a model type, etc. (e.g. <"Rectangle"> , <"My Model 01"> , <"My First Model Type">)

‘:‘ is used for specifying the class of a certain object, or the model where a specific class is included (e.g. <"Red Rectangle 01">:<"Rectangle"> , <"Rectangle">:<"My Model 01">:<"My First Model Type">)

‘>‘ and ‘<‘ in this order are used for filtering the results of a query by a specified class (e.g.: <"B"><<-"requires" >"A"< has as results all objects that fullfill the query criteria <"B"><<-"requires" AND are of class A)

‘->‘ , ‘<-‘ , ‘->>‘ ‘<<-‘ , ‘-->‘ , ‘-->>‘ , ‘<--‘ are used for creating AQL expressions that involve relations in the query critera (will be detailed in the future slides)

AQL Terminal symbols (II)

'{' and '}' are used to represent the object with the specified name (e.g.: {"Rectangle"})

'[' and ']' are used to introduce a criteria to an AQL expression ([?"Radius">"10"])

'(' and ')' are used for deciding the order in which logical operators are evaluated i.e. 'a OR b AND c' and ' (a OR b) AND c ' return different results

'?' is used for imposing a condition on an <u>attribute</u> in the query criteria (e.g.: <"A">[?"Description" like "*OK*"] returns all objects of class A, whose attribute „Description" contains the word „OK")

'!' is used for imposing a condition on a <u>variable</u> during the simulation (e.g.: (<"A">[!"objectCount">"10"]) OR (<"B">[!"objectCount"<="10"]) returns all objects of class A, if the variable object Count is higher than 10 and all objects of class B otherwise.

AQL Non-terminal symbols and key words (I)

Names of classes, names of objects, names of relations, names of attributes, names of variables and constants are denoted by inverted commas (e.g.: "A" , "Rectangle" , "requires" , "Colour")

<Class> ::= '<'*class_name*'>'

Represents a class (e.g. <"requires">)

If the class is not included in the current model, the class has to be denoted explicitly through model name and model type

<Class>':'<ModelName>':'<ModelType>

(e.g.: <"Rectangle">:<"My Model 01">:<"My First Model Type">)

<Object> :: = '<'*object_name*'>'

Represents an object within a concrete model (e.g. <"Red Rectangle 01">)

Should the name of the objects be ambiguous, the name of the class has to be appended to the object's name: <Object>':'<Class>

(e.g.: <"Red Rectangle 01">:<"Rectangle">)

if the object referenced is not part of the current model, the object has to be denoted explicitly through model name and model type:

<Object>':'<Model name>':'<Model type>

(e.g.: <"Red Rectangle 01">:<"My Model 01">:<"My First Model Type">)

<Object>':'<Class>':'<Model name>':'<Model type>

(e.g.: <"Square 01">:<"Square"><"My Model 02">:<"My First Model Type">)

AQL Non-terminal symbols and key words (II)

<Relation> ::= ‘<‘*relation_name*‘>‘

represents a relationclass (e.g. <“requires“>)

<Attribute> ::= ‘<‘*attribute_name*‘>‘

represents the name of an attribute (e.g. <“Color“>, <“Description“>)

<Value> ::= <Constant> | '!' <Variable> | '?' <Attribute>

a value is either a constant, a variable preceded by the symbol ‘!‘ or an attribute preceded by the symbol ‘?‘

<Operator> ::= '>' | '>=' | '=>' | '=' | '<=' | '=<' | '<' |'!='| 'like' | 'unlike'

'like' and 'unlike' are used for alphanumerical signs

? and * are wildcards:„*“ substitutes for any zero or more characters and „?“ substitutes for any one character or less (123??? will match 12313 or 1233, but not 1239919991)

the other operators are used for comparing numerical values

<LogicalOperator> ::= 'AND' | 'OR' | 'DIFF‘

'AND‘ : the result is the intersection set of the two expressions

'OR‘ : the result is the union set of the two expressions

'DIFF' : the result is the difference of the two expressions

'AND' links more strongly than 'OR' and 'OR' more strongly than 'DIFF'

AQL Non-terminal symbols and key words (II)

<AQL expression> ::= '{' [<Object>] [',' <Object>] '}'

the result of an AQL expression is a set of objects (0,1 or more)

<AQL expression> ::= '(' <AQL expression> ')'

expressions may contain parentheses

<AQL expression> ::= <AQL expression> {<LogicalOperator> <AQL expression>}

expressions can be linked to one or more expressions by logical operators

<AQL expression> ::= <AQL expression> '>'<Class>'<'

expression results can be filtered by a specific class

AQL Statements (I)

'<' <class> '>'

the result is all objects of the specified class

Example:

```
<"A">
<"Square":"SecondModel001":"My Second Model Type">
<"A"> [?"Name" like "????e?"]
<"B"> <- "requires"
```

AQL Statements (II)

- <AQL expression> '->' | '<-' | '->>' | '<<-' <Relation>
  - The result contains all objects which are linked through the he given relation with at least one object from the AQL expression
  - '->' returns all direct targets of the relation
  - '<-' returns all direct start objects of the relation
  - '->>' returns all transitive targets of the relation
  - '<<-' returns all transitive start objects of the relation

Example:

- {"A1"}->"requires "
- {"A4"}<-"requires"
- <"A">->"requires"
- <"A"><-"requires"
- {"Rectangle01"}<-“owns“
- ({"A2"}->>"requires") -> "has list"
- {"A4"}<<-"requires“
- <"Rectangle"><<-"owns“
- <"A">->"requires" >"B"<

AQL Statements (III)

<AQL expression>   '->' | '<-'   '<' <Relation> '>'

The result contains all connectors of the specified relation which have as start or target object one of the objects in the AQL expression

'->' returns all connectors originating from the objects of the AQL expression

'<-' returns all connectors ending in the objects of the AQL expression

Please note similarities and differences with before: if you use the '<' and '>' symbols, the result contains the connectors and if you don't use them, it contains the objects

Example:

```
<"B">-><"requires">
<"A"><-<"requires">
{"List003"} <- <"has list">
{"A1"} -> <"has list">
```

AQL Statements (IV)

<AQL expression> '-->' | '-->>' <Attribute>

The result contains all objects which are referenced in the specified attribute of any of the objects in the AQL expression

The ‘-->>’ operator returns is all objects which are transitively referenced in the specified attribute of any of the objects in the AQL expression

Example:

```
<"A"> --> "IsRunBy"
<"A"> -->> "IsRunBy"
{"A5"} --> "IsRunBy"
{"A5"} -->> "IsRunBy"
{"A5"} -->> "IsRunBy" >"Rectangle"<
```

Statements (V)

<AQL expression> '<--'

The result contains all objects which refer any of the objects in the AQL expression

Example:

<"Rectangle"> <--

AQL Statements (VI)

- <AQL expression> '[' <Value> <Operator> <Value> ']'
  - The result contains all objects, whose attributes fulfill the defined criteria
  - Constants (numbers, strings) can only be at the right of the operator
  - To the left of the operator there are only attributes or variable references
  - Note: Queries with variable references as dynamic components in the performer assignment are only allowed in the simulation

Example:

- (<"A"> [?"Description" like ""]) AND (<"A"> [?"A_cost" >=10])
- <"A">[?"Description" like "*Test*"]
- (<"Rectangle">[?"Name" like "M*"]) AND (<"Rectangle">[?"Area" <= 20])
- <"A">[?"Name" like "????e?"]

AQL Statements (VII)

<AQL expression> '['<Value>']' '['<Value> <Operator> <Value>']'

The result contains all objects of the start query where their record attribute or attribute profile fulfills the defined criteria

The first value specifies the name of the record attribute or attribute profile.

See above the rules for the second expression

Note: In case of a record attribute, the criteria is always fulfilled, if at least a table row of the record attribute meets the defined criteria.

Example:

record attribute: <"List">[?"Classification"][?"State" = "Authorized"]

attribute profile: <"A"> [?"Availability"][?"Days per week" >= 3]