VISUALISATION EXPRESSION

3. EXTERNAL COUPLING ADOXX FUNCTIONALITY

Expressions

AdoScript vs. Expressions

AdoScript	Expressions
• Allows embedding external functionality	• No external functionality
• Read and write access to most attributes	• Read access to most attributes, write access only to own attribute
• Must be triggered explicitly by the user	• Are triggered automatically
• Can embed Expressions	• N/A
• Can not be changed by the modeler	• Can be changed by the modeler if not defined as “fixed”
• Usually synchronous execution	• Can be synchronous or asynchronous (idle-processing)
• Any complexity	• Usually less complex than AdoScripts
	• Careful with closed models (values can be outdated)

3 Types of Expressions

- **LeoExpressions:**

  Provide a basic set of functions and operators

  Support for calculation of values, manipulation of strings and other basic operations

  Used inside LEO based languages

- **CoreExpressions:**

  Extension of LeoExpressions

  Only used in EXPRESSION attributes

- **AdoScriptExpressions:**

  Extension of LeoExpressions

  Additional functions can be created (using the keyword FUNCTION)

  Only used in AdoScripts

Expressions – Operations (1)

Logical Op.	`AND, OR, NOT`	Boolean expressions
Comparison Op.	`< > <= >= = <> !=`	Bigger, smaller, equal, diverse
Arithmetic Op.	`+ - * / - (unary)`	
String Op.	`s + t`	Concatenation of Strings s and t
	`n * s`	Replication: String s is replicated n-times
	`s / t`	Count: how often can String s be found in t
	`s SUB i`	The i-th character in String s
	`LEN s`	Length of Strings s

Expressions – Operations (2)

Conversion Op.	`STR val`	String representation of Value val
	`VAL str`	Numerical representation of Strings str

	`CMS measure`
	`PTS measure`
		Conversion of a Unit (in cm or points) to a real number (e.g.: CMS 3.5cm → 3.5).

	`CM real`
	`PT real`
		Conversion of a real number to a Unit (in cm or points; e.g.: CM 3.5 → 3.5cm).

	`uistr(val, n)`	Conversion of a real number to a string in the local format (OS) with n digits.
	`uival( str )`	Conversion of a String in the local format (OS) to a real number.
Sequence Op.	`,`	The comma is used to define a sequence of expressions. The result is always the value of the last expression.

Expressions – Predefined functions (1)

Arithmethic Functions	`abs(x)`
	`max(x, y) min(x, y)`
	`pow(x, y) sqrt(x)`
	`exp(x)`
	`log(x)    log10(x)`
		Arithmetic functions

	`sin(x) cos(x) tan(x)`
	`asin(x) acos(x) atan(x)`
	`sinh(x) cosh(x) tanh(x)`
		Trigonometric functions

	`random()`	Random value 0 >= n < 1
	`round(x)`	Round-to-nearest, i.e. if decimal >= 0.5
	`floor(x) ceil(x)`	Round up/down

Expressions – Predefined functions (2)

String-func.	`search(source, pattern, start)`	Searches in *source* for *pattern*, starting at *start* (0-based), returns index or -1
	`bsearch(source,pattern,start)`	Search begins at end of source string (backwards)
	`copy(source, from, count)`	Copies *count* characters from *source* beginning at *from* (0-based)
	`replall(source, pattern, new)`	Replaces all occurrences of *pattern* in *source* with *new*
	`lower(source)`	Transforms to lower-case
	`upper(source)`	Transforms to upper-case
	`mstr(string)`	Puts the string between “” and escapes special characters

Expressions – Predefined functions (3)

List Funct	`tokcnt(source[,sep])`	Counts tokens in *source* separated by *sep* (default = single whitespace)
	`tokcat(source1, source2 [,separator])`	Concatenates two lists
	`tokunion(source1, source2[, separator])`	Union of two lists
	`tokisect(source1, source2 [, separator])`	Intersection of two lists
	`tokdiff(source1, source2 [, separator])`	Difference of two lists
Color Funct	`rgbval(colorname)`	24bit RGB-Value of the color (by name)
	`rgbval(r, g, b)`	Calculates the RGB-Value for the provided color values.

Expressions – Control structures

Expressions	`set(var, expr)`	*Expr* will be stored in *var*. Variable *var* is created implicitly.
	`cond(cond1, expr1, ..., expr_else)`	Evaluate *cond1*, if true return *expr1*, if false return next condition or return *expr_else*.
	`while(cond, loopexpr[, resultexpr])`	While *cond* is true, evaluate *loopexpr*. Return *resultexpr*.
	`fortok(varname, source, sep, loopexpr [, resultexpr])`	For each element in the list *source*, evaluate *loopexpr*. The current element is stored in *varname*. The list elements are separated by *sep*. Return *resultexpr*.

Expressions – Error handling, type checks

Error handling	`try(expr, failexpr)`	Returns *expr*, if it succeeds, otherwise returns *failexpr*.
Type check	`type(expr)`	Returns the type of the expression. Possible values: "string", "integer", "real", "measure", "time", "expression„ or "undefined„.

Expressions in AdoScript

Types of expressions

Core Expressions:

- Are used to define attributes with the type EXPRESSION
- Can access functions for Core Expressions

AdoScript Expressions:

- Are used in AdoScript
- Can be externalized in functions
- Can access externalized function (defined through keyword FUNCTION)

Core Expressions

Functions for Core Expressions

- The following functions can be used in Core Expressions

```
aval()
avalf()
maval()
paval()
pavalf()
irtmodels()
irtobjs()
profile()
ctobj()
cfobj()
conn()
rcount()
row()
rasum()
prasum()
allobjs()
aql()
prevsl()
nextsl()
asum()
amax()
awsum()
pmf()
class()
mtype()
mtclasses()
mtrelns()
allcattrs()
alliattrs()
allrattrs()
```

- Additionally all LEO expressions and functions can be used

Core Expressions

Attributes of the type EXPRESSION

- An expression attribute contains both a formula and the calculated value
- There are two modes for using expression: fixed and editable
- Fixed expressions store the formula in the default value of the attribute
- An error massage will be returned, if an error occurs when evaluating a formula.
- The last valid result is returned, if an inter-model expression can not be evaluated (when trying to access a not loaded model)
- Expression attributes are always evaluated when an event occurs which can change the value. The changes are shown directly in the user interface

Core Expressions

Attributes of the type EXPRESSION: Definition of expressions as an attribute

Syntax

```
ExprDefinition:    EXPR type:ResultType
                   [format:FormatString]
               expr:[fixed:]CoreExpression
       ResultType :            double | integer | string | time
```

Example

```
EXPR type:string expr:("\"Name = \" + aval(\"Name\")")
```

AdoScript Expressions Application

Expressions can be used directly as arguments of calls and be embedded directly in AdoScript code.

Parenthesis are used to delimit the arguments of an expression.

```
SET n:(copy (vn, 0, 1) + ". " + nn)

IF ( cond( type( n ) = "integer", n = 1, 0 ) )
{
      ...
}

EXECUTE ("SET n:(" + n + ")")
```

Expressions can also be moved to dedicated functions so they can be reused.

AdoScript Expressions

Functions in AdoScript

It is possible to define LEO expressions as reusable functions through the keyword FUNCTION.

Syntax

```
FunctionDefinition  ::= FUNCTION functionName[:global]
                        { FormalFuncParameter }
                        return:expression .
FormalFuncParameter ::= paramName:TypeName .
TypeName            ::= string | integer | real | measure |
                        time | expression | undefined .
```

Example

```
FUNCTION helloWorld world:string
    return:("Hello " + world + "!")

SET hello:(helloWorld("world"))
CC "AdoScript" INFOBOX (hello)
```