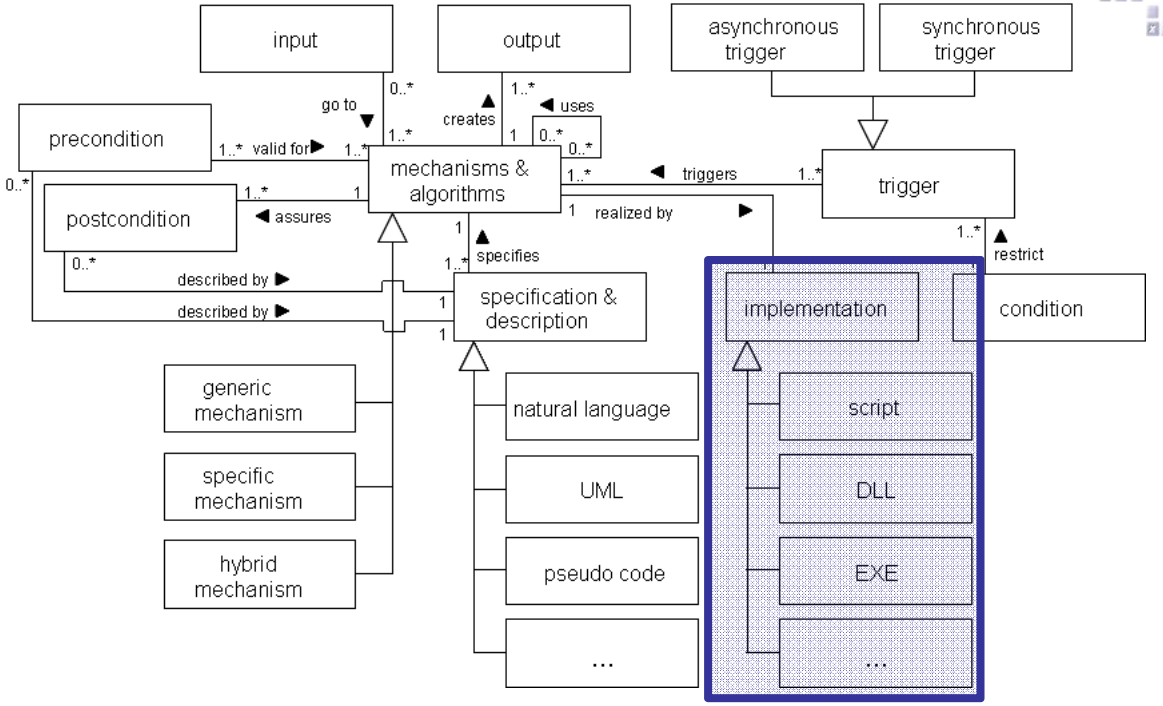
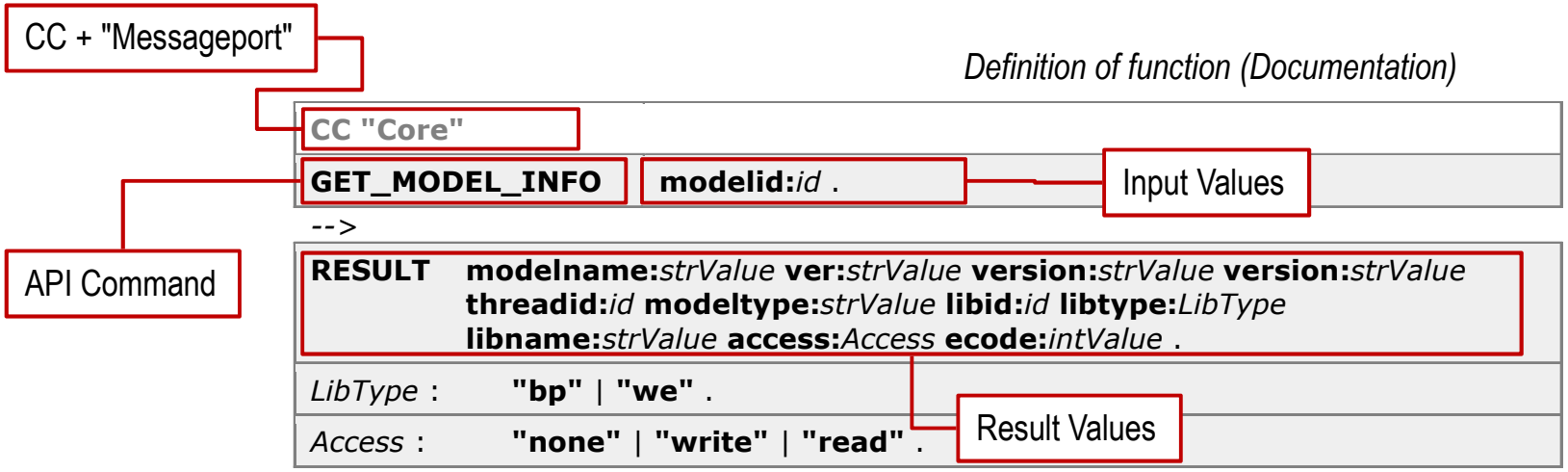


ADOSCRIPT LANGUAGE CONSTRUCTS

API Command Structure

Example of script command- Read of the model information



Code Example

```
# Reading out of the ModelID of a model currently open
CC "Modeling" GET_ACT_MODEL
# Errorcheck ecode
IF (ecode = 0) {
 # Command Call(Keywords in Capitals)
 CC "Core" GET_MODEL_INFO modelid:(VAL modelid)
 # Handling of Return Values
 CC "AdoScript" INFOBOX ("The active model is \"" + modelname + "\" (" +modeltype
+ ")")
} ELSE {
  # ecode returned
  CC "AdoScript" ERRORBOX "No model is opened!"
}
```

AdoScript Basics

Variable declaration

`SET, LEO`

Control structures

`IF/ELSIF/ELSE, WHILE, FOR, BREAK, EXIT, PROCEDURE, FUNCTION`

External programs / File callings (AdoScript, EXE, DLL)

`EXECUTE, SYSTEM, START, CALL`

Sending of messages to ADOxx Component (Messageports)

`CC, SEND`

LEO Expressions

Usage of expressions for call parameters.

AdoScript Operators

Logical

```
AND, OR, NOT
```

Comparison

```
< > <= >= = <> !=
```

Arithmetical

```
+ - * / - (unary)
```

Strings

```
s + t, n * s, s / t, s SUB i, LEN s
```

Convertings

```
STR value, VAL string
```

AdoScript Functions

Arithmetical

```
abs(x), max(x, y), min(x, y), pow(x, y), sqrt(x),
exp(x), log(x), log10(x)
```

Strings

```
search(source,pattern,start),
bsearch(source,pattern,start),
copy(source,from,count),
replall(source,pattern,new), lower(source),
upper(source)
```

Lists

```
token(source,index[,separator]),
tokcnt(source[,separator]),
tokcat(source1,source2[,separator]),
tokdiff(source1,source2[,separator]),
tokisect(source1,source2[,separator]),
tokunion(source1,source2[,separator]),
```

AdoScript: Procedure Concepts

```
ProcedureDefinition :      PROCEDURE [global]
ProcedureName
[ MainParameter ] { FormalProcParameter }
{ StatementSequence } .
MainParameter :      TypeName:paramName .
FormalProcParameter :
      paramName:TypeNameOrReference .
TypeNameOrReference :      TypeName | reference .
TypeName :    string | integer | real | measure | time
|
array | expression | undefined .
ProcedureName :      keyword .
ProcedureCall :      anyLeoElement .
```

```
PROCEDURE MYPROC integer:n val:string result:reference
{
   SET result:(val + STR n)
}
```

AdoScript: Functions concept

FunctionDefinition ::= FUNCTION functionName[:global]
{ FormalFuncParameter }
return:expression .

FormalFuncParameter ::= paramName:TypeName .

TypeName ::= string | integer | real | measure |
time | expression | undefined .

Example:

```
FUNCTION fak n:integer
    return:(cond (n <= 1, 1, n * fak (n - 1)))

SET m:(fak (10))
```

Expressions in AdoScript

Expressions can be used direct as arguments in calls.

Use closures () in order to delineate arguments of an expression.

Example

```
SET n:(copy (vn, 0, 1) + ". " + nn)
IF ( cond( type( n ) = "integer", n = 1, 0 ) )
{
      ...
}
EXECUTE ("SET n:(" + n + ")")
```

Summary of AdoScript Language Elements

AdoScript language elements

Command execution

EXECUTE	SEND
CC	SYSTEM
START	CALL

Allocation elements

SET	SETG
SETL	

Control elements

IF	ELSIF
ELSE	WHILE
FOR	BREAK
NEXT	EXIT

Definition of Procedures/Functions

PROCEDURE	FUNCTION

LEO (Return Format) Handling

LEO	LEO parse

StatementSeq :	{ *Statement* } .
Statement :	*Execute* | *Send* | *CC* | *System* | *Start* | *Call* | *Set* | *SetL* | *SetG* | *Leo* | *IfStatement* | *WhileStatement* | *ForStatement* | *BreakStatement* | *ExitStatement* | *FunctionDefinition* | *ProcedureDefinition* | *ProcedureCall* .
Execute :	*ExecuteFile* | *ExecuteEx* .
*Execute*File :	**EXECUTE file:***scriptText* [**scope:***ScopeSpec*] .
ExecuteEx :	**EXECUTE** *scriptText* [**scope:***ScopeSpec*] .
ScopeSpec :	**separate | same | child** .
Send :	**SEND** *msgText* **to:***msgPortName* [**answer:***varName*] .
CC :	**CC** *msgPortName* [**debug**] [**raw**] *anyLeoElement* .
System :	**SYSTEM** *strExpr* [**with-console-window**] [**hide**] [**result:***varName*] .
Start :	**START** *strExpr* [**cmdshow:***CmdShow*] .
CmdShow :	**showmaximized** | **showminimized** | **showminnoactive** | **shownormal** .
Call :	**CALL dll:***strExpr* **function:***strExpr* { *InputParam* } [**result:***varName*] [**freemem:***strValue*] .
InputParam :	**varName:***anyExpr* .
Set :	**SET** { *VarAssignment* } .
SetL :	**SETL** { *VarAssignment* } .
SetG :	**SETG** { *VarAssignment* } .
VarAssignment :	**varName:***anyExpr* .
Leo :	**LEO** [*parseCmd*] { *accessCmd* } .
parseCmd :	**parse:***stringExpr* .
accessCmd :	**get-elem-count:***varName* | **set-cur-elem-index:***intExpr* | **get-keyword:***varName* | **is-contained:***varName* [**:***strExpr*] | **get-str-value:***varName* [**:***strExpr*] | **get-int-value:***varName* [**:***strExpr*] | **get-real-value:***varName* [**:***strExpr*] | **get-tmm-value:***varName* [**:***strExpr*] | **get-time-value:***varName* [**:***strExpr*] | **get-modifier:***varName***:***strExpr* .
IfStatement :	**IF** *booleanExpr* **{** *StatementSequence* **}** { **ELSIF** *booleanExpr* **{** *StatementSequence* **}** } [**ELSE {** *StatementSequence* **}**] .
WhileStatement :	**WHILE** *booleanExpr* **{** *StatementSequence* **}** .
ForStatement :	*ForNumStatement* | *ForTokenStatement* .
ForNumStatement :	**FOR** *varName* **from:***numExpr* **to:***numExpr* [**by:***numExpr*]

AdoScript Programm Guidelines

- If your are programming AdoScript, please consider following rules:
  - Files which contain AdoScript should be named file.asc
  - The returning result of a message port command should be under the command.

```
GET_CLASS_NAME classid:intValue .
  # --> RESULT ecode:intValue classname:strValue isrel:intValue
```

  - Indent a block with two spaces
  - Don‘t use the tabulator to indent blocks
  - Decompose the complexity of the programm by using procedures and functions.