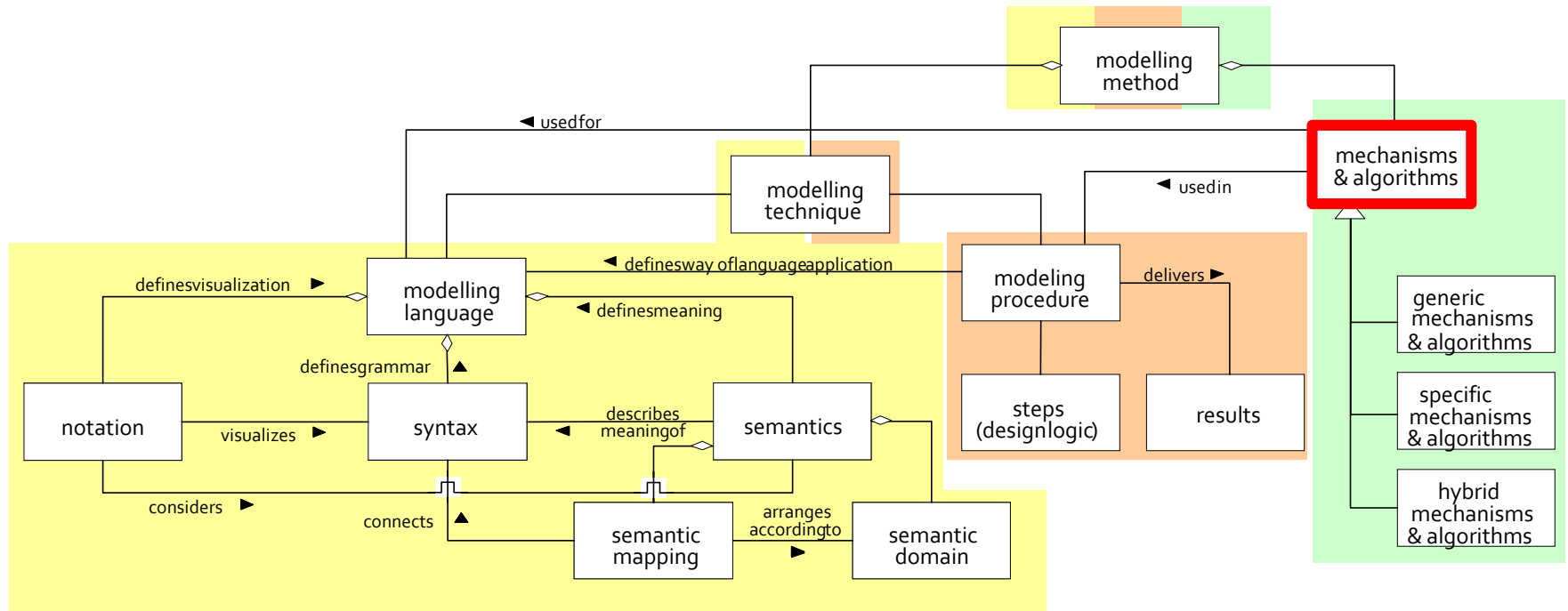


AdoScript

Generic Modelling Method Framework



Reference: Karagiannis, D., Kühn, H.: „Metamodelling Platforms“. In Bauknecht, K., Min Tjoa, A., Quirchmayer, G. (Eds.): Proceedings of the Third International Conference EC-Web 2002 – Dexa 2002, Aix-en-Provence, France, September 2002, LNCS 2455, Springer, Berlin/Heidelberg, p. 182 ff.

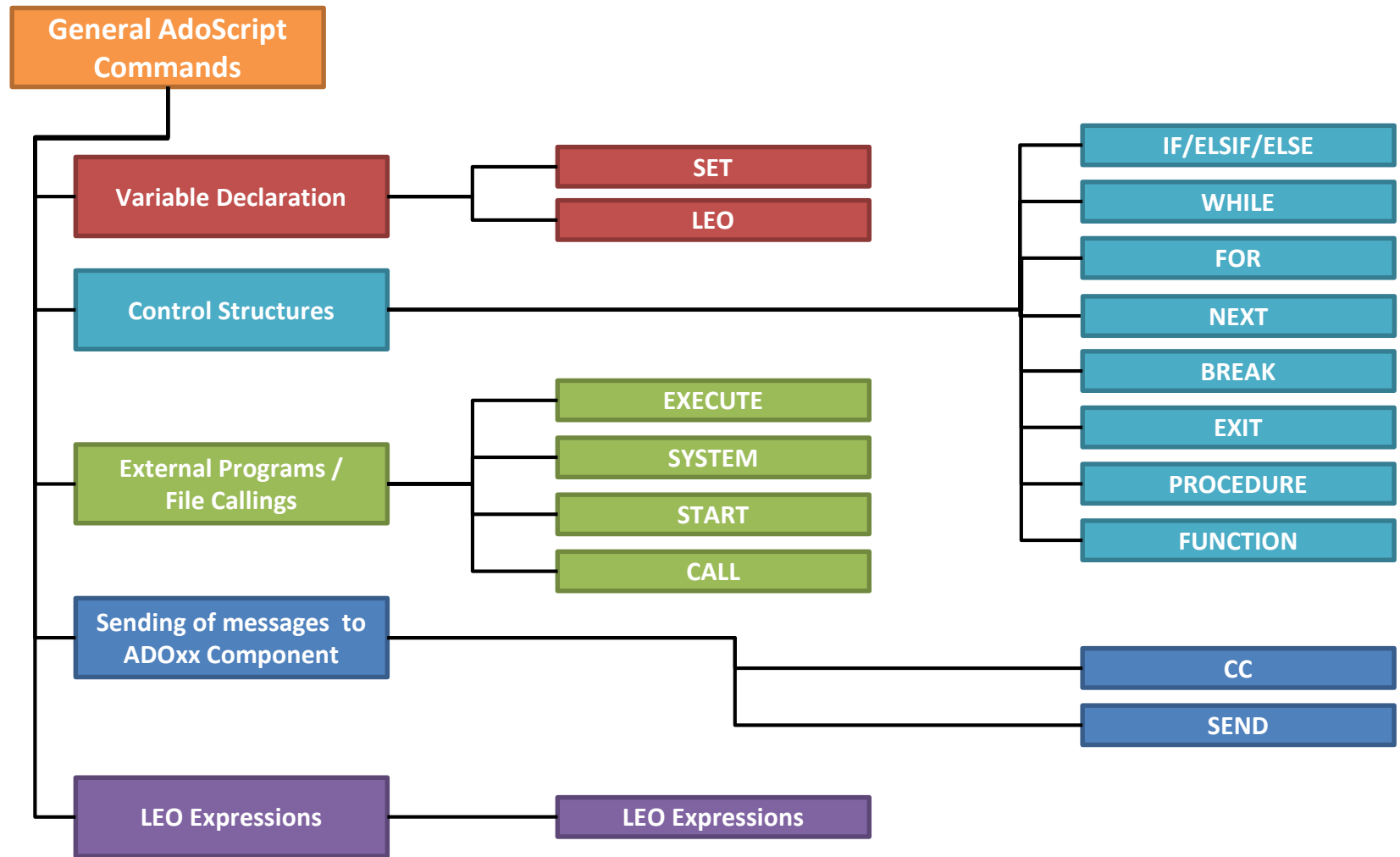
AdoScript vs Expression Attributes

AdoScript	Expression Attributes
<ul style="list-style-type: none">• Allows embedding external functionality	<ul style="list-style-type: none">• No external functionality
<ul style="list-style-type: none">• Read and write access to most attributes	<ul style="list-style-type: none">• Read access to most attributes, write access only to own attribute
<ul style="list-style-type: none">• Mostly triggered explicitly by the user	<ul style="list-style-type: none">• Are triggered automatically
<ul style="list-style-type: none">• Can embed Expressions	<ul style="list-style-type: none">• N/A
<ul style="list-style-type: none">• Typically can not be changed by the modeler	<ul style="list-style-type: none">• Can be changed by the modeler if not defined as “fixed”
<ul style="list-style-type: none">• Usually synchronous execution	<ul style="list-style-type: none">• Can be synchronous or asynchronous (idle-processing)
<ul style="list-style-type: none">• Any complexity	<ul style="list-style-type: none">• Usually less complex than AdoScripts
	<ul style="list-style-type: none">• Careful with closed models (values can be outdated)

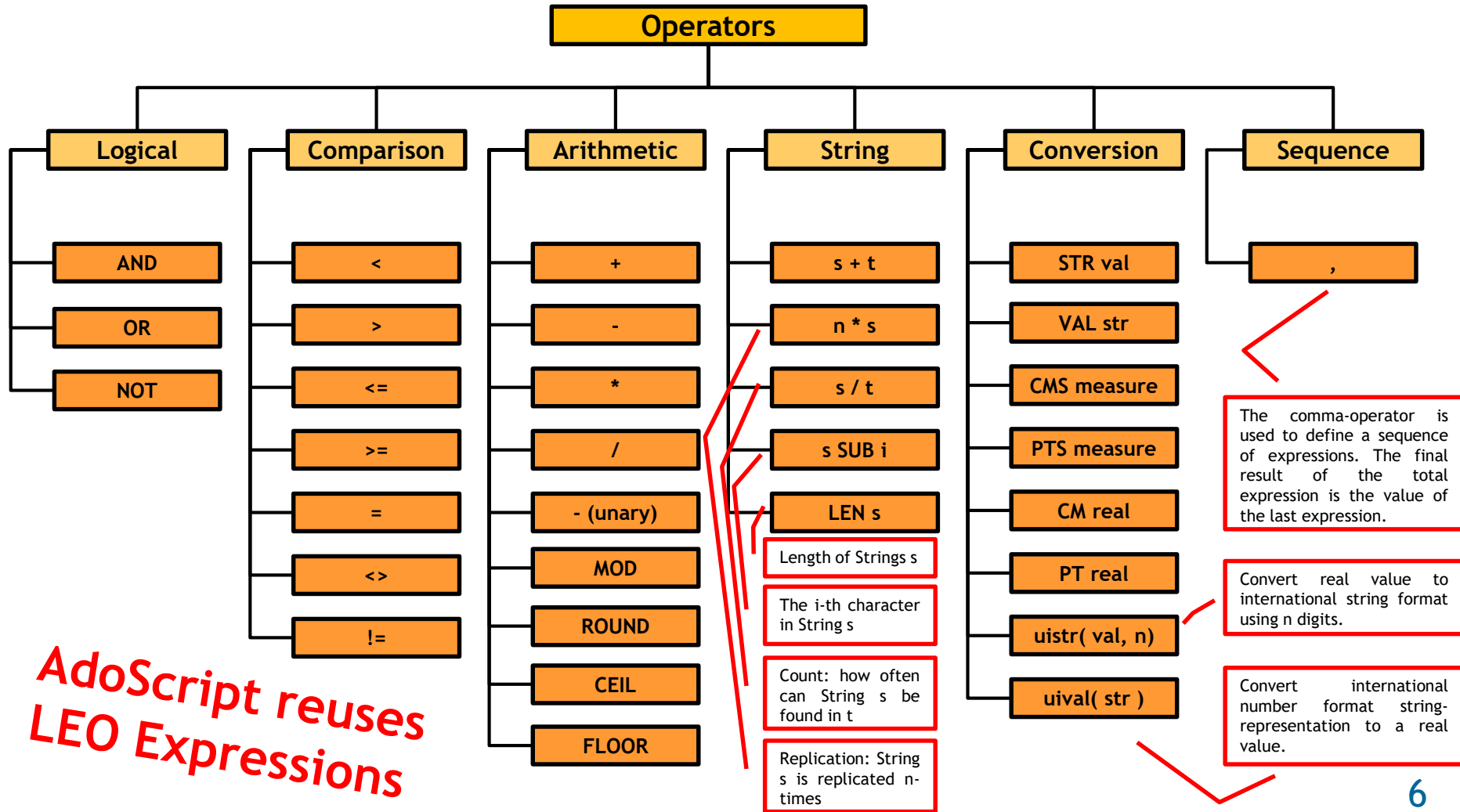
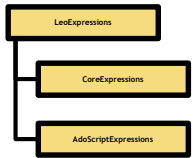
AdoScript Introduction

- AdoScript is a DSL for ADOxx
- It allows you to create algorithms which are executed on the model
- AdoScript is based on LEO
- What AdoScript does not have:
 - A debugger
 - Object Orientation
 - Complex Type System (important available types are: Numeric, String, Measure, Maps and Arrays)
- Exercises use the "SmartCityLibrary-I-ModellingExercises.abl" library and "SmartCity-I-SmartParking.adl" models

Structure of General AdoScript Commands



Operators of Expressions

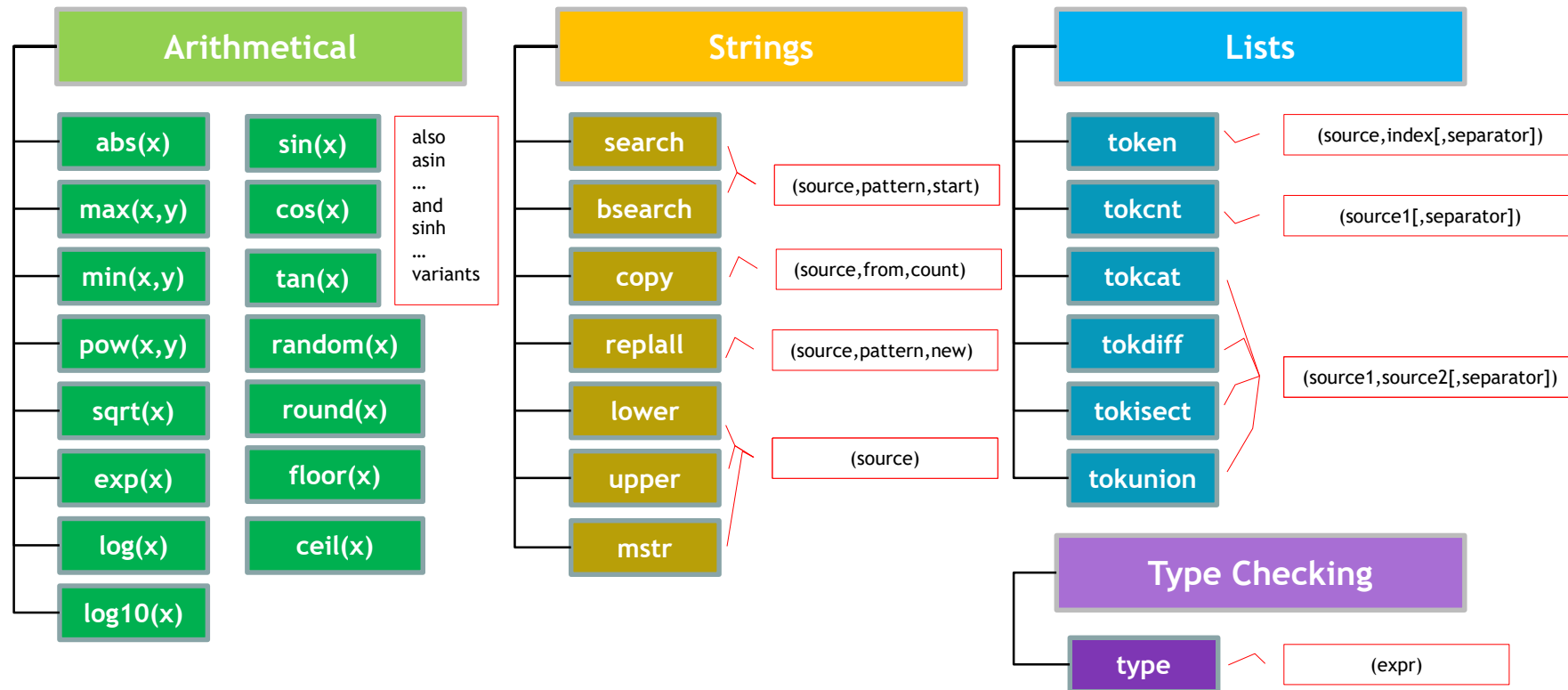


AdoScript reuses
LEO Expressions

Predefined Functions of Expressions

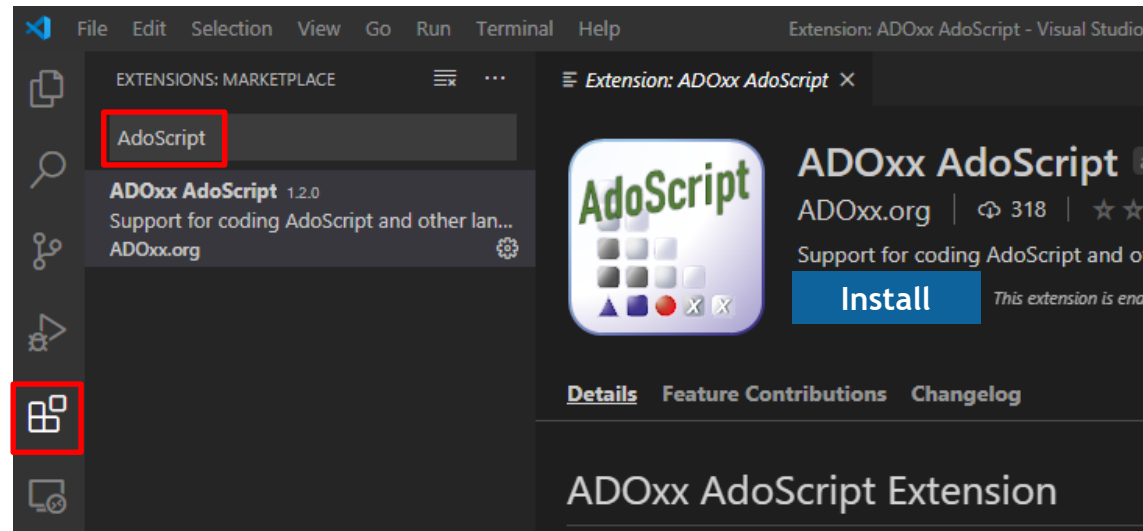
*AdoScript reuses
LEO Expressions*

Some important Functions:



Set Up AdoScript Development Environment

- Download Visual Studio Code
 - <https://code.visualstudio.com/>
- Install VSC AdoScript extension
 - <https://marketplace.visualstudio.com/items?itemName=ADOxxorg.adoxx-adoscript>
 - Alternative:



Set Up a Debug Environment I

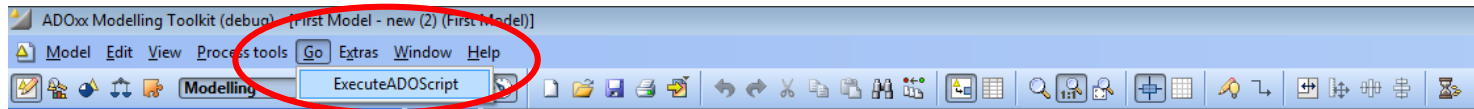
- Create a file which will contain your ADOScript (*.asc)
- Create a Trigger (Menu Item) for executing the Script contained in the file

Library Attribute "External coupling" under "Add-ons" (Dynamic Library)

```
#Menu Item - Execute AdoScript from file
ITEM "ExecuteADOScript" modeling:"Go"
CC "AdoScript" FREAD file:("C:\\Temp\\myAdoScript.asc")
EXECUTE (text)
```

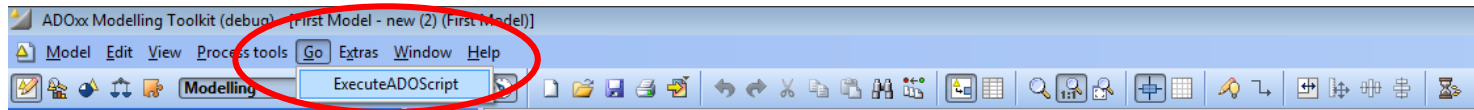
Set Up a Debug Environment I

- We can now execute the AdoScript if we click on the Menu Item which we created



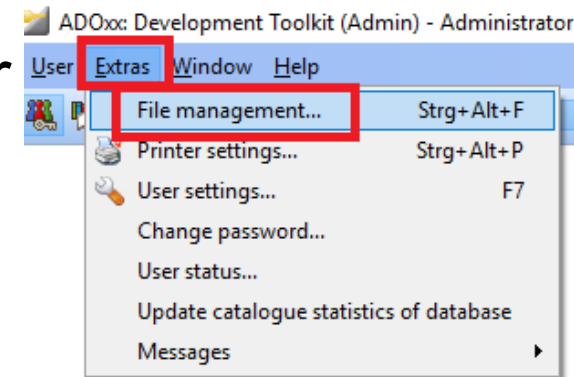
- You can also **directly add the AdoScript code in the External Coupling Section of the Development Toolkit.** However this would **require to restart the Modeling Toolkit** after every change.

Set Up a Debug Environment I



- Be aware that you have to include your final file in your library. To do this go to Extras -> File Management in the Development Toolkit and import your file.
- Thereafter you can access the file in the external coupling directly from the database:

```
#Menu Item - Execute AdoScript from file  
ITEM "ExecuteADOScript" modeling:"Go"  
CC "AdoScript" FREAD file:"db:\\myAdoScript.asc"  
EXECUTE (text)
```



Set Up a Debug Environment II

OR

- Create a Menu Item which opens an Editbox, where you can insert and execute AdoScript Code

Library Attribute
"External coupling"
under "Add-ons"
(Dynamic Library)

```
SmartCity-I-Dynamic Modelling Exercise Library - Library attributes

Modi:
METHOD graphRep:"Method GraphRep" {
  GROUP "Smart City Modelling" {
    MODELTYPE "SmartCity"
  }
  GROUP "Smart City Analysis" {
    MODELTYPE "Analysis-SmartParking"
    MODELTYPE "Analysis-SmartEmergency"
    MODELTYPE "Analysis-SmartTrack"
    MODELTYPE "Analysis-SmartAmbulance"
    MODELTYPE "Analysis-SmartMarathon"
  }
}

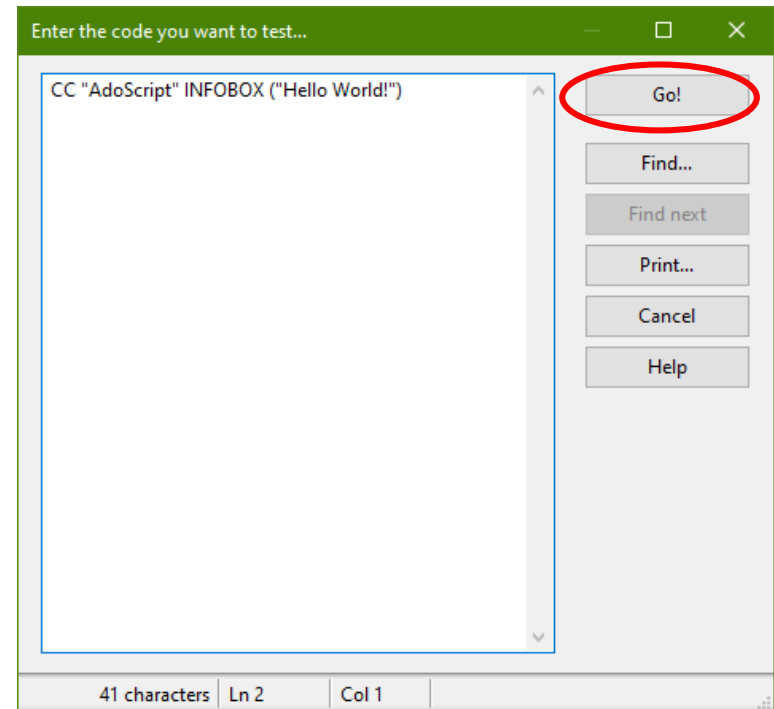
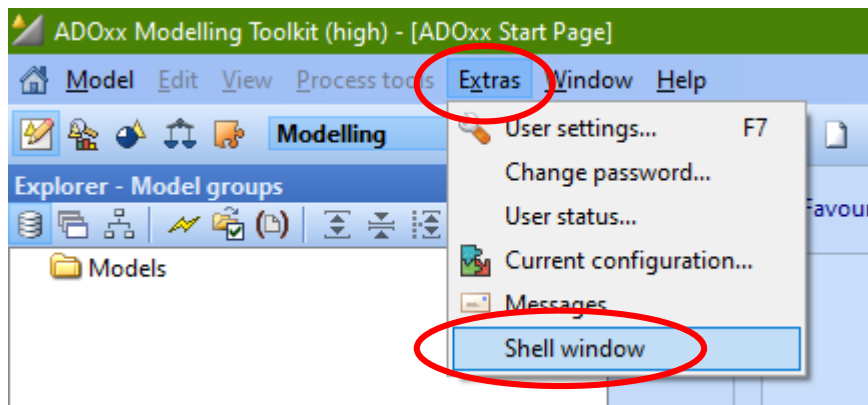
Versioning format:

External coupling:
ITEM "Shell window" modeling:"Extras"
SET endbutton:"ok"
WHILE (endbutton = "ok")
{
  CC "AdoScript" EDITBOX text:(adoscript)
  title:"Enter the code you want to test..."
  oktext:"Go!"
  IF (endbutton = "ok")
  {
    SET adoscript:global:(text)
  }
}
```

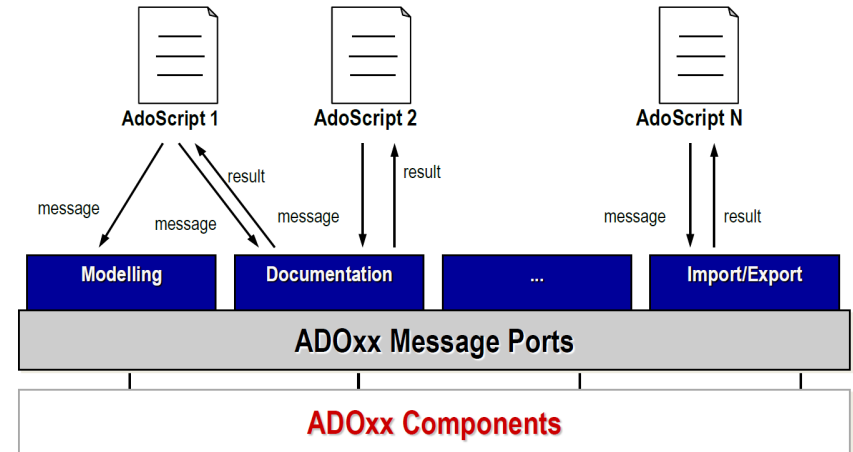
```
#Menu Item
ITEM "Shell window" modeling:"Extras"
SET endbutton:"ok"
WHILE (endbutton = "ok") {
  CC "AdoScript" EDITBOX text:(adoscript)
  title:"Enter the code you want to test..."
  oktext:"Go!"
  IF (endbutton = "ok") {
    SET adoscript:global:(text)
    EXECUTE (text)
  }
}
```

Set Up a Debug Environment II

- We can now execute the AdoScript if we click on the Menu Item which we created, enter the code in the pop-up that appears and select "Go!"



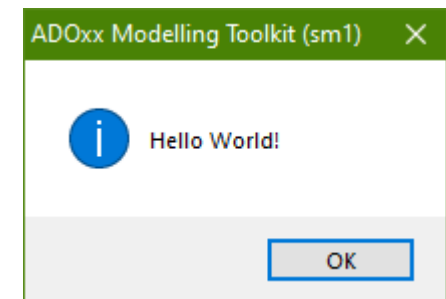
Hello World Example



Call Command Message Port Command Input Value

↓ ↓ ↓ ↓

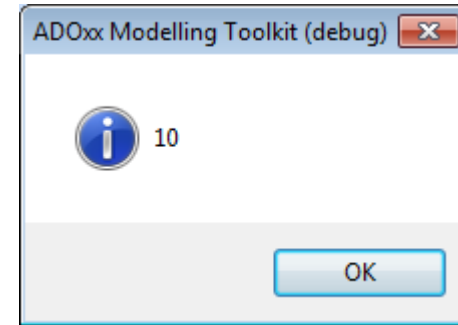
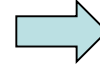
CC "AdoScript" INFOBOX ("Hello World!")



- Use the **Debug-Mode** to find errors
 - Write debug between MessagePort and Command
- ```
CC "AdoScript" debug INFOBOX ("Hello World!")
```

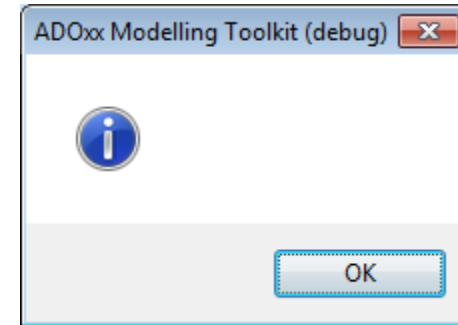
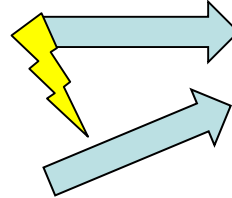
# AdoScript Basic Examples

```
SET b:10
SET c:7
CC "AdoScript" INFOBOX (b)
```



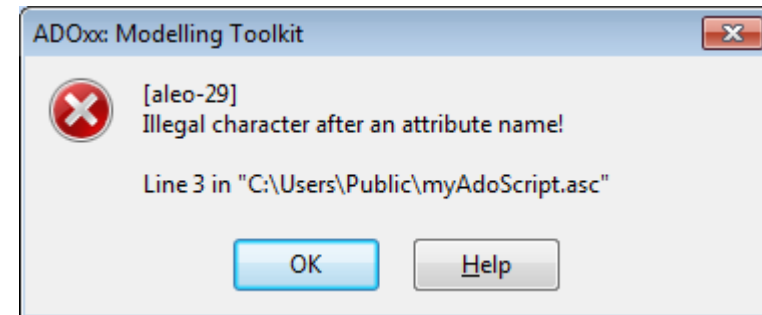
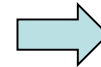
```
SET b:10
SET c:7
CC "AdoScript" INFOBOX b

SET b:10
SET c:7
CC "AdoScript" INFOBOX x
```



```
SET b:10
SET c:7
CC "AdoScript" INFOBOX b+c
```

Use () for  
evaluating b+c

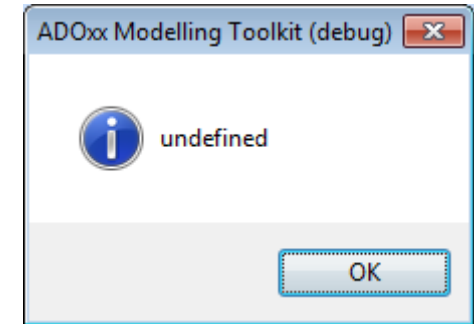
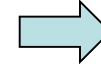


( ) is used to specify an Expression (here: get value of a variable)

# AdoScript Basic Examples

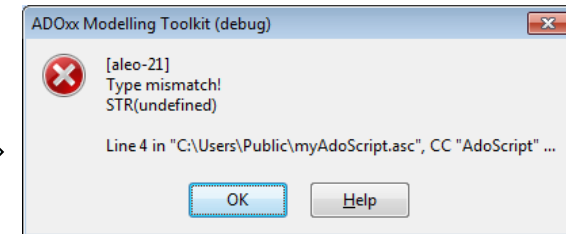
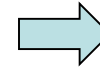
```
SET m:3
SET w:m
CC "AdoScript" INFOBOX (type(w))
```

undefined refers to a variable that has not been initialised

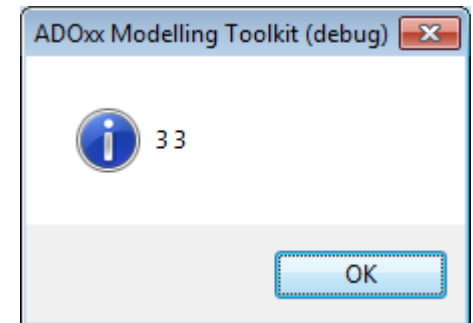
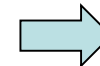


```
SET m:3
SET w:m
CC "AdoScript" INFOBOX ((STR m)+" "+(STR w))
```

SET { VarAssignment } .  
SETL { VarAssignment } .  
SETG { VarAssignment } .  
VarAssignment : LValue : anyExpr .



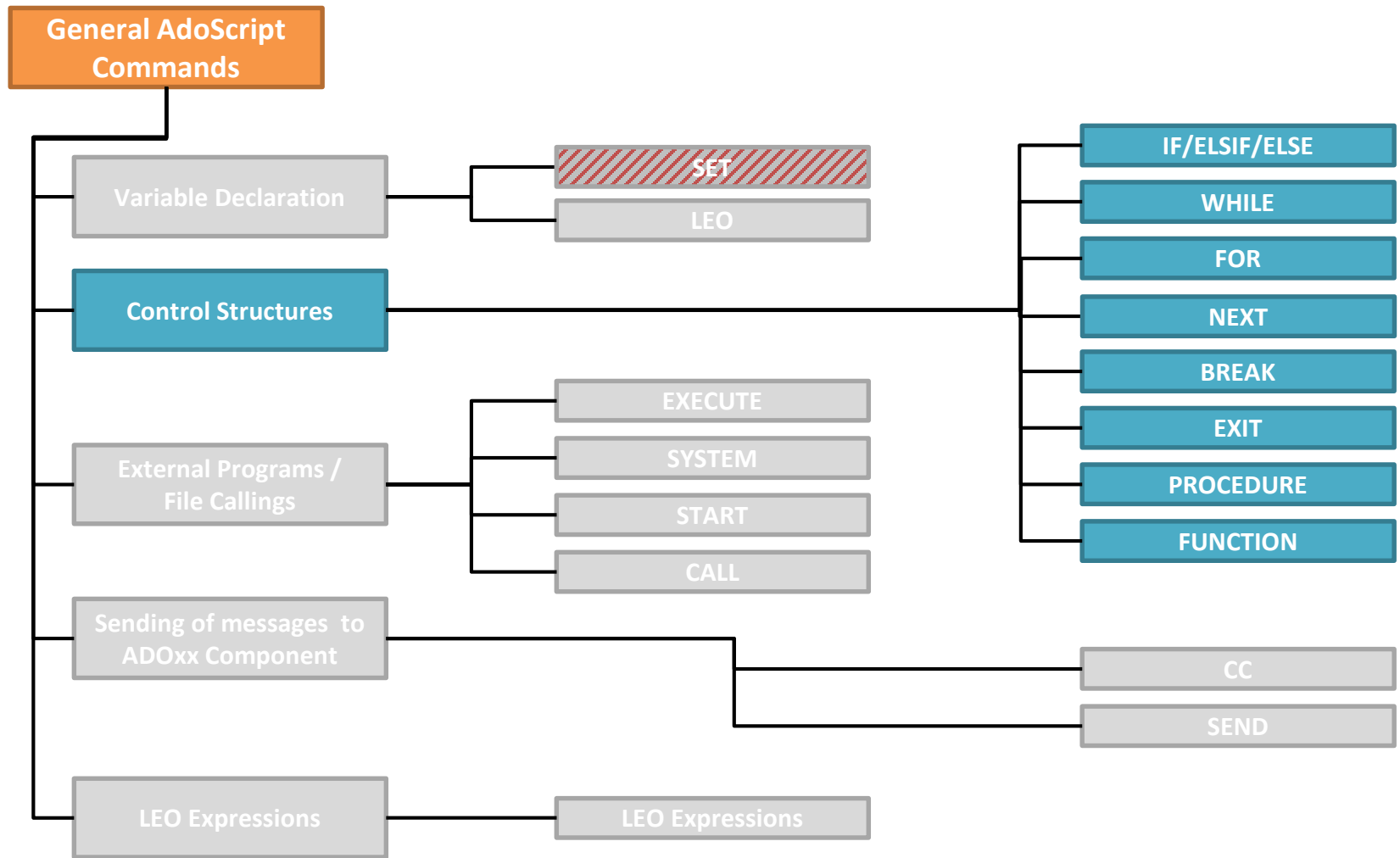
```
SET m:3
SET w:(m)
CC "AdoScript" INFOBOX ((STR m)+" "+(STR w))
```



() is used to specify an Expression



# Structure of General AdoScript Commands



# Conditional

## ■ If/Else Structure

```
SET a:1
SET b:7
```

Setting a variable:  
**SET** *varname: valORexpr*

```
IF (a<2) {
 CC "AdoScript" debug INFOBOX (a)
} ELSIF (a<3) {
 CC "AdoScript" debug INFOBOX (a+b)
} ELSE {
 CC "AdoScript" debug INFOBOX ((a+b)*2)
}
```

# Loops

- **WHILE Loop**

```
SET a:5
```

```
SET b:7
```

```
WHILE (a<b) {
 CC "AdoScript" debug INFOBOX (a)
 SET a:(a+1)
}
```

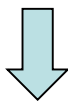
- **FOR Loop (Lists / Tokens)**

```
SET a:5
```

```
SET b:"1 3 5"  List with three Tokens
```

Convert String to Value

```
FOR localVar in:(b) {
 CC "AdoScript" debug INFOBOX ((VAL localVar)+a)
}
```



# Loops and other Control Commands

- **FOR Loop – Numeric**

```
SET a:5
FOR localVar from:1 to:5 by:2 {
 CC "AdoScript" debug INFOBOX (localVar + a)
}
```

- **NEXT**

- Skips all following commands of the loop and executes the WHILE / FOR loop again

- **BREAK**

- Leaves a WHILE / FOR loop, without executing the loop again

- **EXIT**

- Stops AdoScript / Procedure execution

# Exercise

- Create a Script which prints out all numbers between 1 (incl.) and 10 (excl.) except 5

**Hint:** CC "AdoScript" debug INFOBOX ("Ausgabe")

# Exercise – Pseudo Code

- Create a Script which prints out all numbers between 1 (incl.) and 10 (excl.) except 5

Hint: CC "AdoScript" debug INFOBOX ("Ausgabe")

- Pseudo Code:
  1. Set a counter variable
  2. While the counter is smaller than 10
    1. If the counter is not 5 → Print counter
    2. Increment counter

# Exercise – Solution

- Create a Script which prints out all numbers between 1 (incl.) and 10 (excl.) except 5

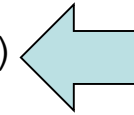
**Hint:** CC "AdoScript" debug INFOBOX ("Ausgabe")

```
SET counter:1

WHILE ((counter)<10) {
 IF ((counter)<>5) {
 CC "AdoScript" debug INFOBOX (counter)
 }
 SET counter:((counter)+1)
}
```

# Defining own Functions

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

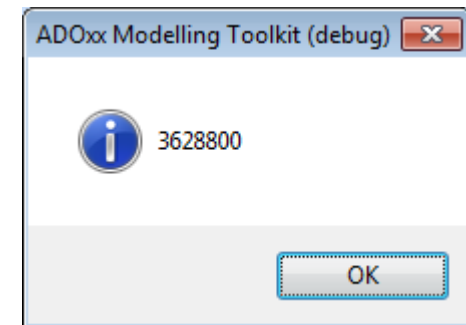


Expression



Recursive

```
CC "AdoScript" debug INFOBOX (fak(10))
```

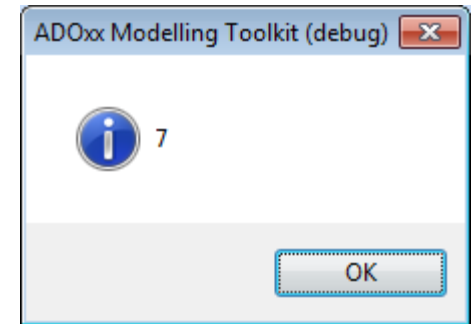




# Defining own Procedures

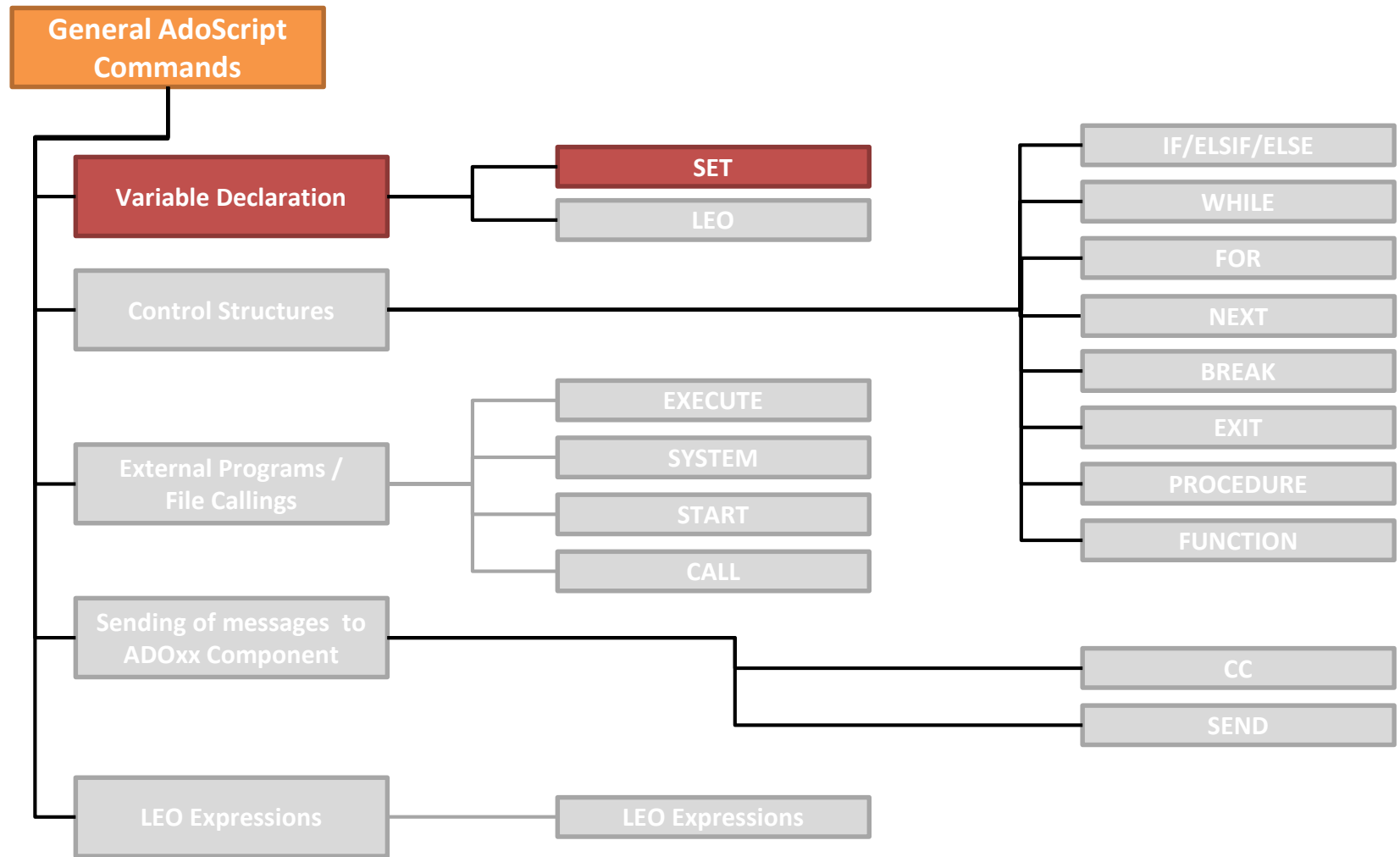
```
PROCEDURE APROC a1:integer a2:integer r:reference
{
 SET r:(a1+a2)
}
```

```
SET result:1
APROC a1:3 a2:4 r:result
CC "AdoScript" debug INFOBOX (result)
```



- Difference Procedure vs Function:
  - Body of functions are an Expression,  
Body of procedures are AdoScript
  - A function returns one value,  
A procedure can return several values through changing  
provided references

# Structure of General AdoScript Commands

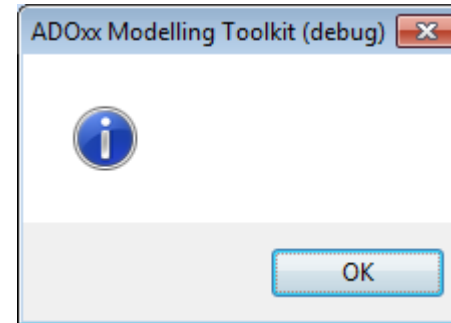


# Scope

- **SET** assigns values to **new or existing AdoScript runtime variables**
- A variable's lifetime usually is the **execution of the current AdoScript or procedure**.
- Using **SETG** lets the variable exist for the **whole ADOxx session, so it can be reused in succeeding executions of the same or other AdoScripts**.
- If a variable exists in the current scope, **SET overwrites its value**.
- If a variable should generally be **local, also if a same-name variable already exists in a higher scope, SETL has to be used**.

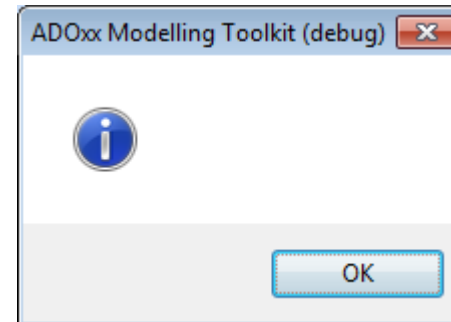
# Scope

```
SCOPE_TEST
CC "AdoScript" INFOBOX (b)
EXIT
```



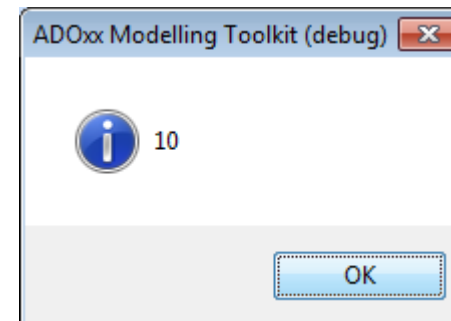
```
PROCEDURE SCOPE_TEST {
 SET b:10
}
```

```
SCOPE_TEST
CC "AdoScript" INFOBOX (b)
EXIT
```



```
PROCEDURE SCOPE_TEST {
 SETL b:10
}
```

```
SCOPE_TEST
CC "AdoScript" INFOBOX (b)
EXIT
```



```
PROCEDURE SCOPE_TEST {
 SETG b:10
}
```

# Scope

```
Declares the global variable 'a' and initiates it with '0'
SETG a:0
CC "AdoScript" INFOBOX (STR a) # Prints '0'
```

```
SCOPE_TEST
```

```
CC "AdoScript" INFOBOX (STR a) # Prints '1'
```

```
PROCEDURE SCOPE_TEST {
```

```
 # Changes the value of the global variable 'a' to '1'
```

```
 SET a:1
```

```
 CC "AdoScript" INFOBOX (STR a) # Prints '1'
```

```
 # Declares the local variable 'a' and inits it with '2'
```

```
 SETL a:2
```

```
 CC "AdoScript" INFOBOX (STR a) # Prints '2'
```

```
 # Changes the value of the local variable 'a' to '3'
```

```
 SET a:3
```

```
 CC "AdoScript" INFOBOX (STR a) # Prints '3'
```

```
}
```

shadowing

Local variable  
is accessed

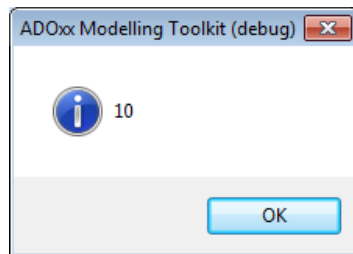
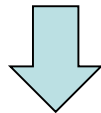
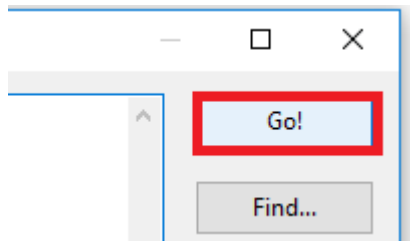
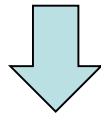
# Scope

1.

```
Declares the global variable 'a' and
inits it with ,10'
```

```
SETG a:10
```

```
CC "AdoScript" INFOBOX (STR a)
```

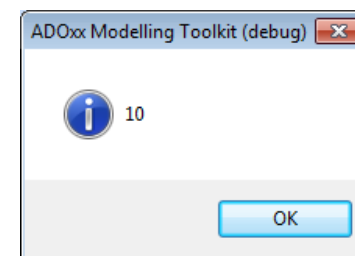
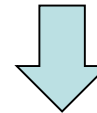
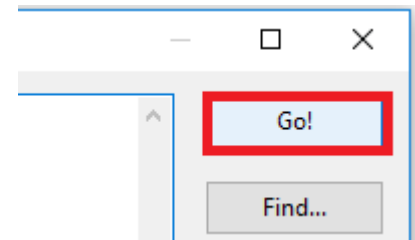
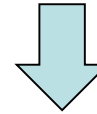


SETG values  
are kept after  
execution!  
(until the tool  
is closed)

2.

```
Prints the variable 'a' as
defined globally
```

```
CC "AdoScript" INFOBOX (STR a)
```



# Scope

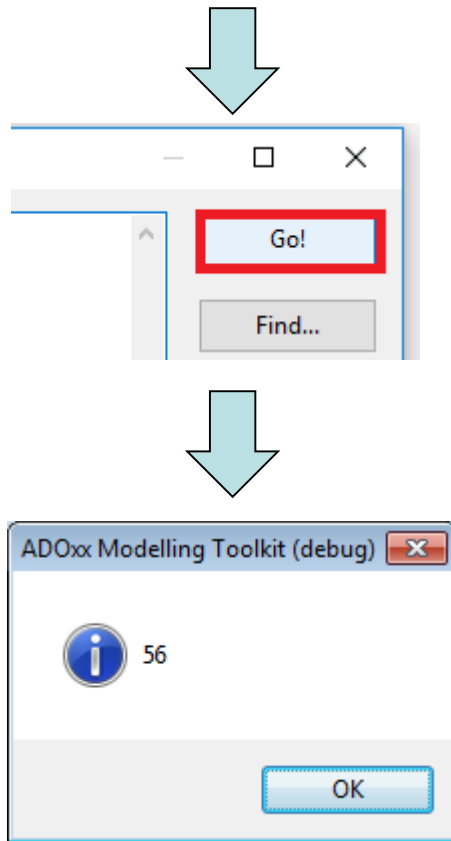
3.

4.

```
Declares the local variable 'a' and
initiates it with ,56'
```

```
SETL a:56
```

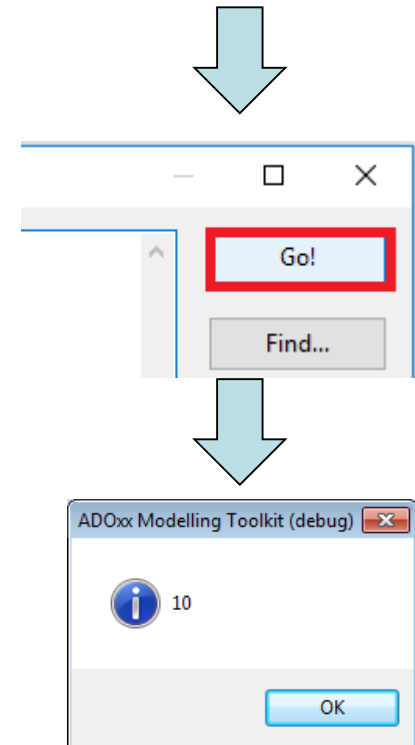
```
CC "AdoScript" INFOBOX (STR a)
```



SETG values  
are kept after  
execution!  
(until the tool  
is closed)

```
Prints the variable 'a' as
defined globally
```

```
CC "AdoScript" INFOBOX (STR a)
```



# Lists

- Lists are heavily used in AdoScript
  - They are represented as a string containing values (tokens) separated with a blank (default)

```
SET a:"1 3 5" # List with three values: 1, 3 and 5
```

```
FOR i in:(a) {
 CC "AdoScript" debug INFOBOX (i)
}
```

Lists are a string following a specific structure, not a special data type

- Further functions

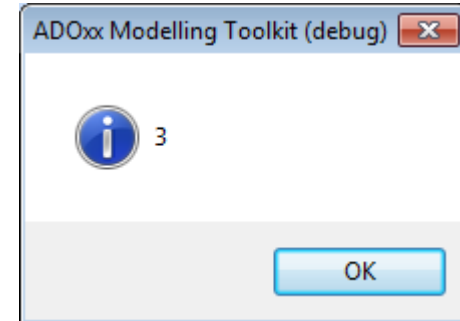
**token**(source,index[,separator])  
**tokcnt**(source[,separator])  
**tokcat**(source1,source2[,separator])  
**tokdiff**(source1,source2[,separator])  
**tokisect**(source1,source2[,separator])  
**tokunion**(source1,source2[,separator])



# Lists - Examples

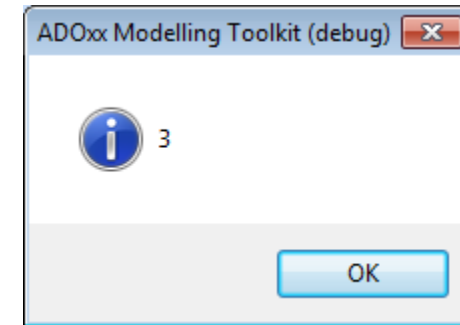
```
SET a:"1 3 5"
```

```
CC "AdoScript" debug INFOBOX (token(a,1)) →
```



```
SET a:"1 3 5"
```

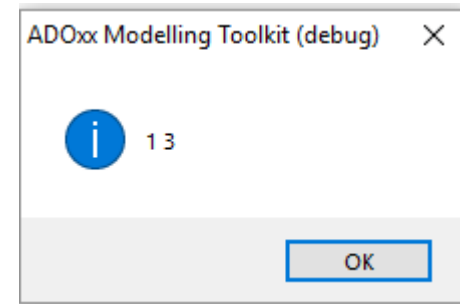
```
CC "AdoScript" debug INFOBOX (tokcnt(a)) →
```



```
SET a:"1 3 5"
```

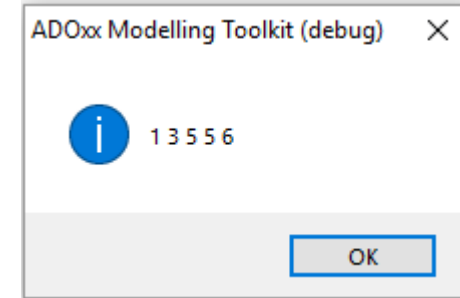
```
SET a:(tokdiff(a,"5 6"))
```

```
CC "AdoScript" debug INFOBOX (a) →
```

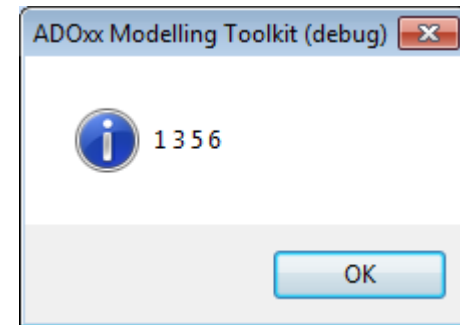


# Lists - Examples

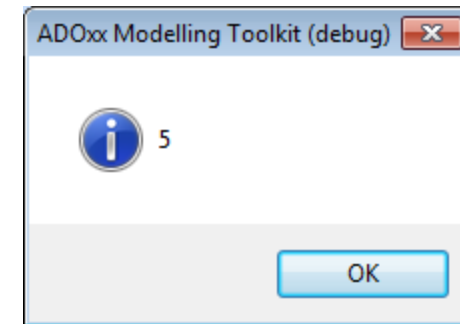
```
SET a:"1 3 5"
SET a:(tokcat(a,"5 6"))
CC "AdoScript" debug INFOBOX (a)
```



```
SET a:"1 3 5"
SET a:(tokunion(a,"5 6"))
CC "AdoScript" debug INFOBOX (a)
```



```
SET a:"1 3 5"
SET a:(tokisect(a,"5 6"))
CC "AdoScript" debug INFOBOX (a)
```



# Arrays

- Arrays are used less often
  - Usually commands return lists containing tokens

```
SET a:({"Bonn", "Köln", "Leverkusen", "Wuppertal"})
SET a[2]:"Siegburg" # replaces "Leverkusen"
SET b:({{4711, 4712}, {6006, 6007, 6008}})
SET b[0, 1]:2011 # replaces 4712 with 2011
SET b[1]:({7755, 7756}) # replaces {6006, 6007, 6008}
SET b[0][1]:2012 # syntax error!
 # [][] just allowed in expressions
```

#Access

```
CC "AdoScript" INFOBOX (a[1])
```

# Maps

## ■ Create a map

```
SETL emptyMap:(map())
SETL mMap:({ 1780: "Sim", 1920: "Sala", 2007: "Bim" })
CC "AdoScript" INFOBOX ("This is the initial map: " + STR mMap)
```

Key

Value

## ■ Remove item

```
SETL dummy:(merase (mMap, 1780))
```

SETL is used so that the expression on the right is evaluated

## ■ Add item

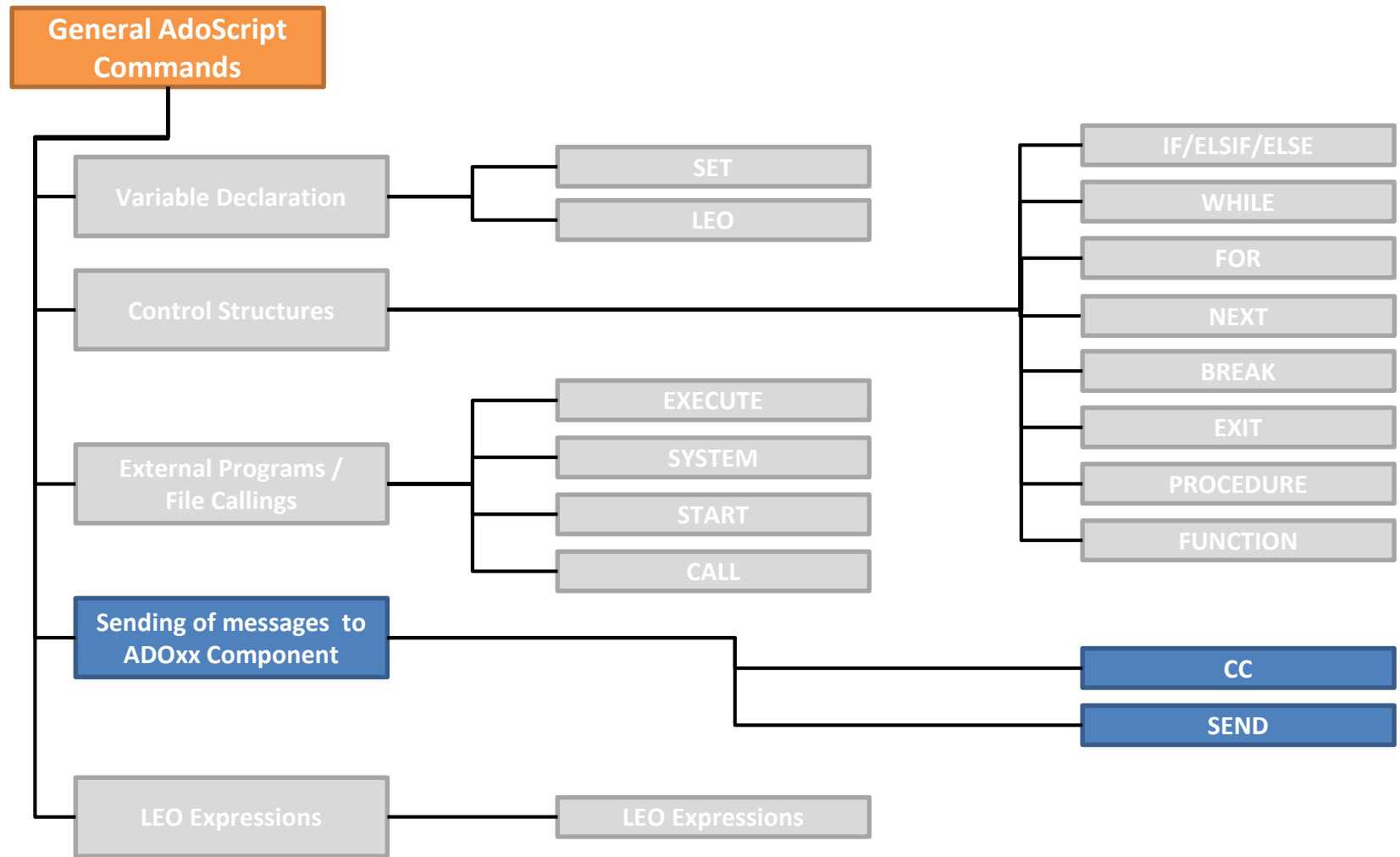
```
SETL mMap[2014]:"Bam"
```

## ■ Get Item

```
SETL value:(mMap SUB 2014)
```

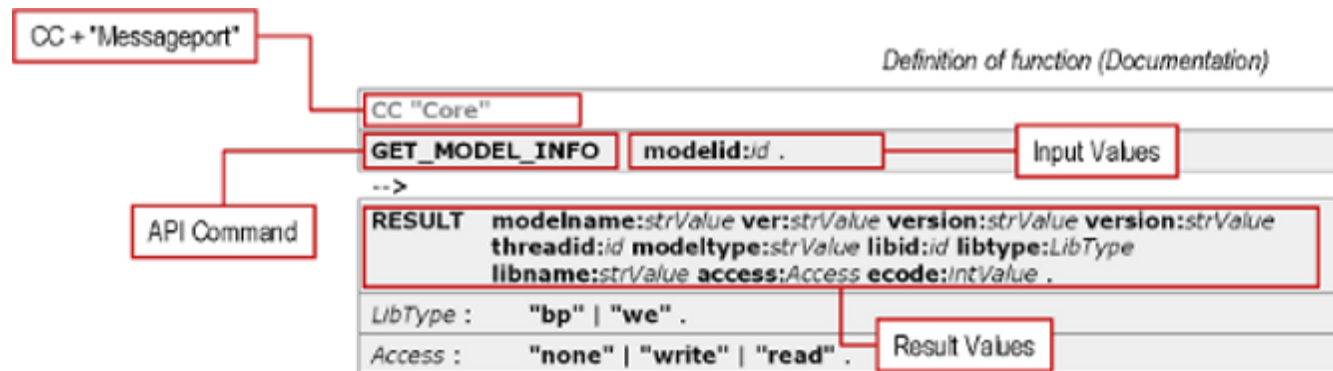
```
SETL value:(mMap[2014])
```

# Structure of General AdoScript Commands



# The CC Command

- CC is used to access the model data, components and functionalities provided by the ADOxx tool through special Message Ports and their API-Commands.



- Results are provided by setting specific variables which can then be accessed.
  - Be careful with overwriting variables, either directly or through CC commands!

# The CC Command – Message Ports

- Some of the available Message Ports are:
  - Core – accessing and manipulating model data at its core
  - AdoScript – some useful dialogues and other general support
  - Modeling – interface to the Modelling Component
  - AQL – for executing AQL queries
  - Analysis – interface to the Analysis Component
  - Simulation – interface to the Simulation Component
  - ImportExport – interface to the Import/Export Component

# Core/Modeling Message Port

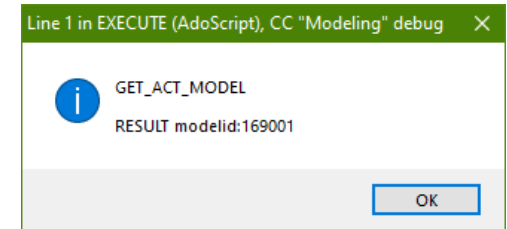
- The Core and Modeling Message Ports are the most important port for accessing your model
  - In this slides we present only an excerpt of important functions
- To create good mechanisms you will need more functions than can be presented here
- Use the ADOxx Help and the AdoScript Documentation:
    - <https://www.adoxx.org/AdoScriptDoc/index.html>



# Core/Modeling Message Port – Example

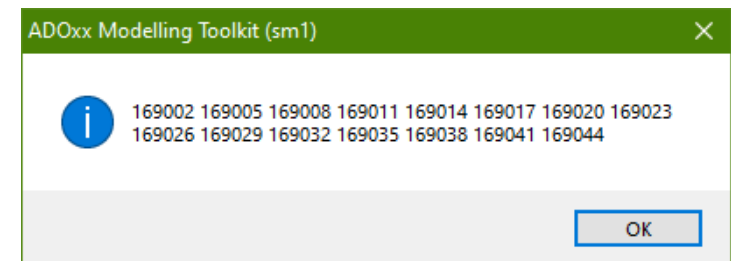
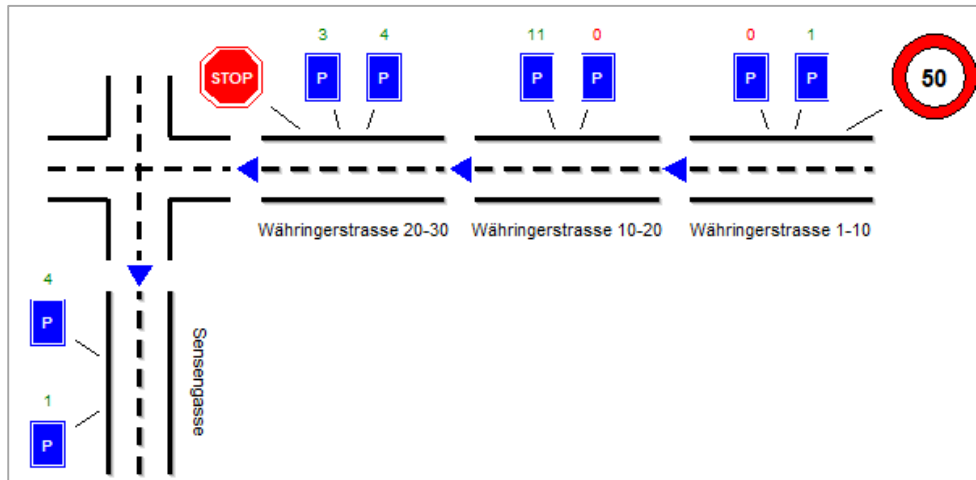
- Get all objects of current model

```
CC "Modeling" debug GET_ACT_MODEL
#--> RESULT modelid: intValue .
```



```
CC "Core" debug GET_ALL_OBJS modelid:(modelid)
#--> RESULT ecode: intValue objids: strValue .
```

```
CC "AdoScript" debug INFOBOX (objids)
```



# Core/Modeling Message Port – Example

- Get the name of the first object

```
CC "Modeling" GET_ACT_MODEL
```

```
#--> RESULT modelid: intValue .
```

```
CC "Core" GET_ALL_OBJS modelid:(modelid)
```

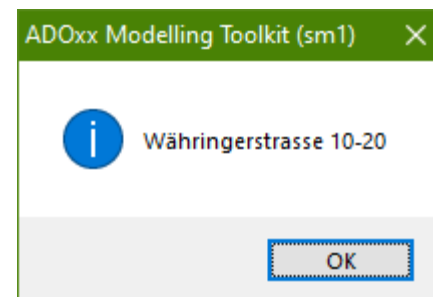
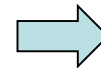
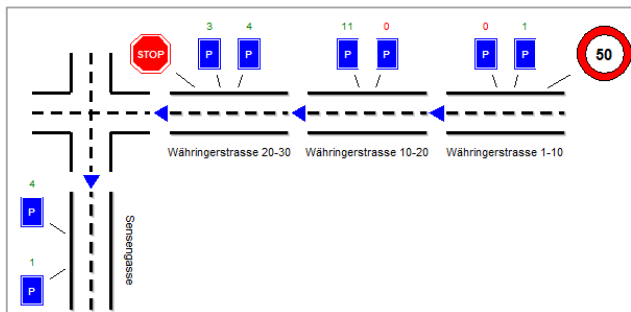
```
#--> RESULT ecode: intValue objids: strValue .
```

Conversion with  
VAL necessary!

```
CC "Core" debug GET_ATTR_VAL objid:(VAL token(objids, 0))
attrname:"Name"
```

```
#--> RESULT ecode: intValue val: anyValue .
```

```
CC "AdoScript" debug INFOBOX (val)
```



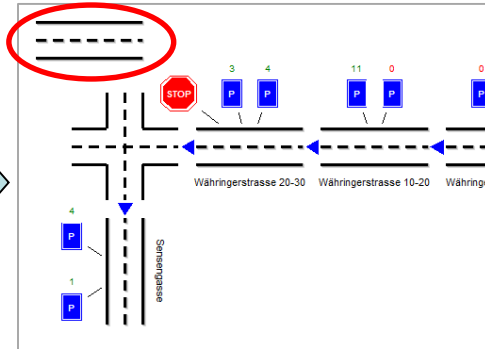
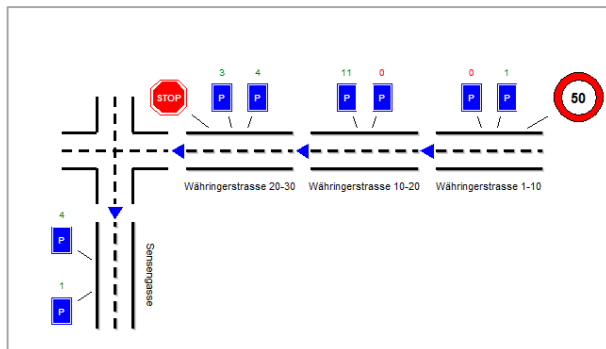
# Core/Modeling Message Port – Example

- Create new object and position it

```
CC "Modeling" GET_ACT_MODEL
#--> RESULT modelid: intValue .
```

Enter a  
classname  
which exists in  
your model

```
CC "Core" GET_CLASS_ID classname: "Road"
#--> RESULT ecod: intValue classid: intValue isrel: intValue
CC "Core" CREATE_OBJ modelid: (modelid) classid: (classid)
objname: "test box"
#--> RESULT ecod: intValue objid: id .
CC "Core" SET_ATTR_VAL objid: (objid) attrname: ("Position")
val: ("NODE x:2cm y:1cm")
```



Note: Such changes are not  
saved automatically!  
Use "Core" SAVE\_MODEL  
or let user save the model.

# Triggering AdoScript execution

- Triggers describe when ADOscript code is fired

Dynamic

Library management

Settings Checks Management

Application libraries:

- ADOxx 1.5 Experimentation Library
- ADOxx 1.5 Dynamic Experimentation Library
- ADOxx 1.5 Static Experimentation Library

Class hierarchy...

Class attributes...

Attribute scopes...

Library attributes...

Predefined analysis queries...

Predefined evaluation queries...

Release library

Close Help

ADOxx 1.5 Dynamic Experimentation Library - Library attributes

Model:

MODELTYPE "First Model"

INCL "Entity"

INCL "Relation"

INCL "Attribute"

INCL "PER\_Constraint\_Attribute"

INCL "Relation\_Entity"

Versioning format:

External coupling:

```
INIT GLOBAL VARS
ON_EVENT "AppInitialized"
{
}

##AdoScript
ITEM "ExecuteAdoScript" modeling:"-Go"
OC "AdoScript" FREAD file:("C:\\Users\\Public\\myAdoScript.as")
```

Description

Add-ons

Modeling

Analysis

Simulation

Library Attribute "External coupling" under "Add-ons" (Dynamic Library)

Events

```
ON_EVENT "AppInitialized" { #do smthg. }
```

Event examples:

- AfterCreateModelingNode
- AfterCreateModelingConnector
- BeforeDeleteInstance
- AfterEditAttributeValue
- ...

```
ITEM "ADOxx-Standard-Methode"
acquisition:"~Hilfe"
modeling:"~Hilfe"
analysis:"~Hilfe"
#do smthg.
```

Menu Item

# Triggering AdoScript execution

- Execution of AdoScript commands over program call attributes in the notebook of objects

ADOxx-Default-BP-Library 1.0 - Edit class hierarchy

Class hierarchy:

- Activity
  - Conversion\_ Longstring (LONGSTRING)
  - Doku String (STRING)
  - DokuSim String (STRING)
  - SortAttr String (STRING)
  - Accountable for approving results Intermodel reference (INTERREF)
  - Aggregated costs Floating number (DOUBLE)
  - Aggregated execution time Time (TIME)
  - Aggregated personnel costs Floating number (DOUBLE)
  - Aggregated resting time Time (TIME)
  - Aggregated transport time Time (TIME)
  - Aggregated waiting time Time (TIME)
  - Beschreibung String (STRING)
  - Bezeichnung String (STRING)
  - Classification Enumeration list (ENUMERATION)
  - Comment String (STRING)
  - Cooperation/participation Intermodel reference (INTERREF)
  - Costs Floating number (DOUBLE)
  - Description String (STRING)
  - Display responsible role Enumeration (ENUMERATION)
  - EDP batch costs Floating number (DOUBLE)
  - EDP transaction costs Floating number (DOUBLE)
  - External documentation Programcall (PROGRAMCALL)**
  - Info on results String (STRING)
  - Input Intermodel reference (INTERREF)
  - Input/Output Expression (EXPRESSION)
  - IT systems Expression (EXPRESSION)
  - Kommentar String (STRING)
  - Language Expression (EXPRESSION)
  - Modelling direction Expression (EXPRESSION)
  - Number Floating number (DOUBLE)
  - Open questions String (STRING)
  - Order Integer (INTEGER)
  - Organizational unit Intermodel reference (INTERREF)
  - Output Intermodel reference (INTERREF)

ADOxx-Default-BP-Library 1.0 - Extend value range - External docu...

Attribute "External documentation"

Value range:

- Winword
- Excel**
- Powerpnt
- Editor
- Notepad
- Wordpad

Apply

Cancel

Help

Attribute type  
PROGRAMCALL

Edit value range

Programm call:

Excel

Parameters:

file

Default value:

File filter:

\*.xls

AdoScript:

START ("Excel " + file)

Delete

Add

Can also be triggered through GraphRep (see HOTSPOT)

# Triggering AdoScript execution

- ADOScript Shell Window
- Toolbar Icons
  - See `INSERT_ICON` and `SET_ICON_CLICK_HDL` of Message Port "Application" for details
- Context Menu Items
  - See `INSERT_CONTEXT_MENU_ITEM` and `SET_CMI_SELECT_HDL` of Message Port "Application" for details

# Exercise

- Create a Procedure which deletes an object. The parameters of the Procedure are the name of the object (which should be deleted) as well as the class name

#Example Call

```
DELETEOBJ objName:"CrossRoad-12628" className:"CrossRoad"
EXIT
```

Hint: look in the API for  
DELETE\_OBJS

Some AdoScript Documentation:

<https://www.adoxx.org/AdoScriptDoc/index.html>

**THANK YOU FOR YOUR  
ATTENTION**

