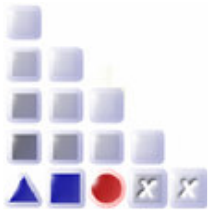


ADOxx 1.5 Cookbook for Implementing Mechanisms and Algorithms

Abstract: The document contains an exercise whose purpose is to help a user new to ADO*xx* to get familiar with the ADO*xx* Development Toolkit. After having implemented the modeling language as described in the “ADOxx 1.5 Cookbook for Implementing Modeling Language”, go through the exercises in this document and implement simple mechanisms and algorithms of the modeling method. Download the ADO*xx* Tutorial slides from the adoxx.org portal.

Table of Contents

1. Preparation

The preparation phase consists in ensuring the prerequisites for implementing mechanisms and algorithms for your modeling method:

- download and installation of the latest version of ADOxx
- download and unpacking the Method Engineering Tutorial Package
- creating an user for modeling and testing purposes
- implementing the modeling language as described in the "ADOxx 1.5 Cookbook for Implementing Modelling Language"

Note: The Tutorial Library used for this Cookbook is the result of the ADOxx 1.5 Cookbook for Implementing a Modelling Language. If you have not finished the ADOxx 1.5 Cookbook for Implementing a Modelling Language yet you can import the Tutorial Library from the folder "<mypath>\Method-Engineering-Package\Tutorial_Library.

Slides for implementing mechanisms and algorithms are located in the file "**3_Mechanisms and Algorithms Implementation_PUBLIC.pdf**", which you can find under "<mypath>\Method-Engineering-Package\Tutorial_Slides\".

2. Insert AdoScript Shell

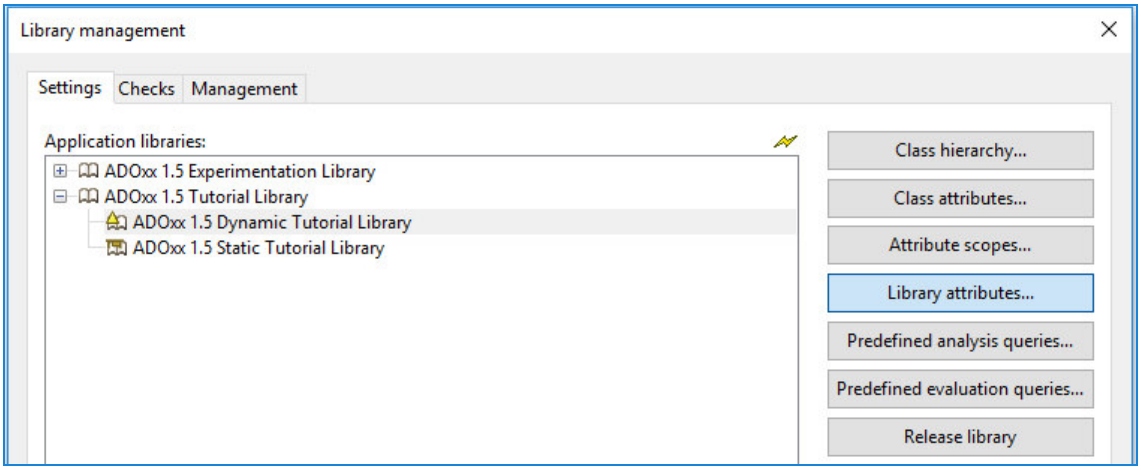
Firstly a new *menu item* 'AdoScript-Shell' in a new menu 'ADOxx Tutorial' should be created. When clicking on the menu item, a new text window should be opened where AdoScript commands can be entered and are executed by clicking the "Run" button.

2.1. Create New Menu Button "AdoScript-Shell"

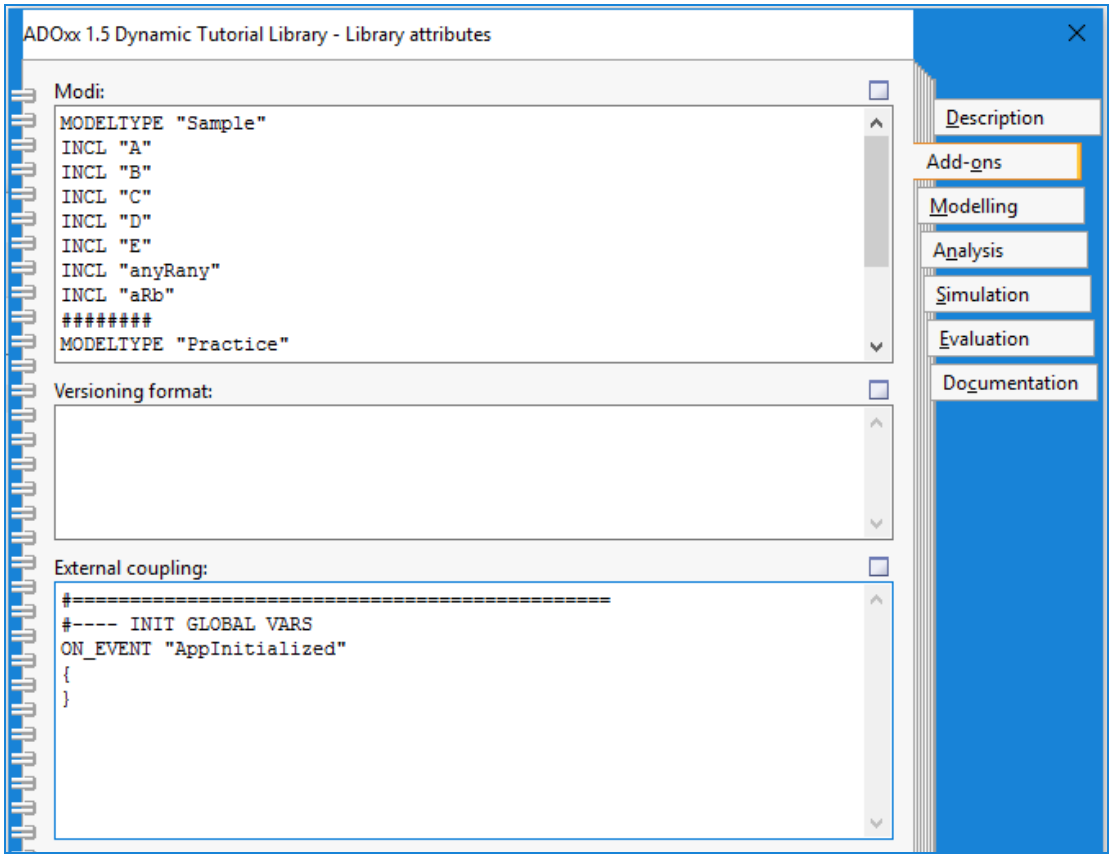
Note: If you are familiar with the implementation of the AdoScript Shell, then you can download the library, which already includes the AdoScript Shell and continue with section **2.2.**

You can find the library in the Method Engineering Tutorial Package under "*<mypath>\Method-Engineering-Package\Tutorial_Libraries\Mechanisms and Algorithms Implementation Libraries*"

1. Open up the "Library Management" section
2. Click the "Library attributes" button



3. Select the "Add-ons" tab
4. Open the "External coupling" library attribute:

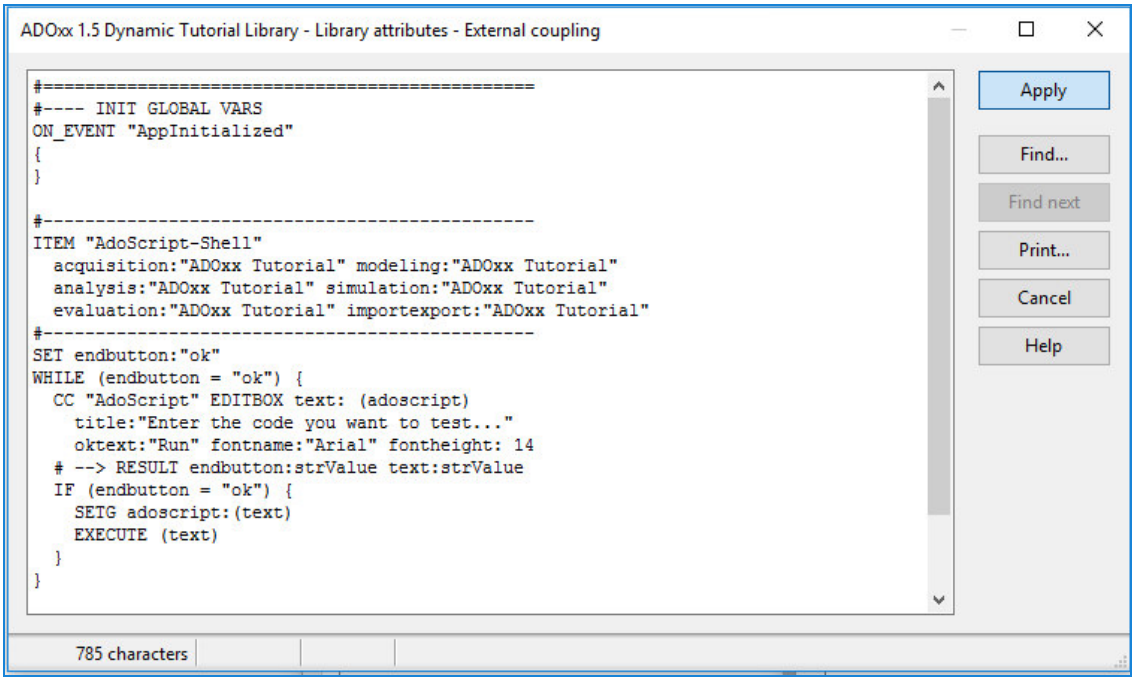


5. insert the following text:

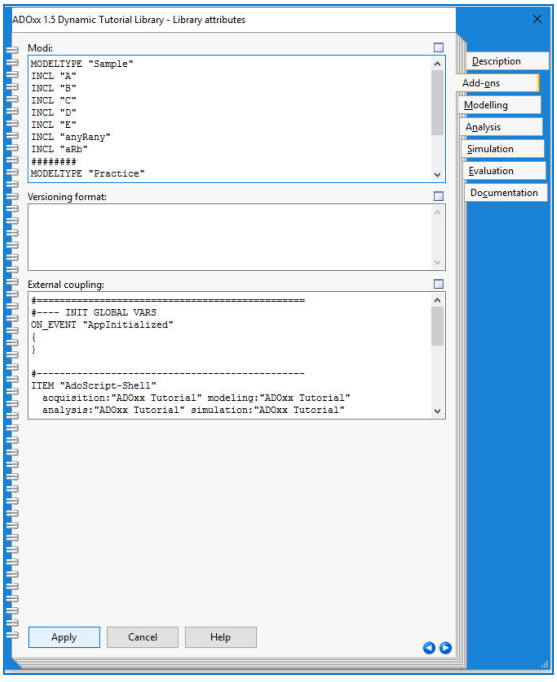
```
#-----------------------------------------------
ITEM "AdoScript-Shell"
  acquisition:"ADOxx Tutorial" modeling:"ADOxx Tutorial"
  analysis:"ADOxx Tutorial" simulation:"ADOxx Tutorial"
  evaluation:"ADOxx Tutorial" importexport:"ADOxx Tutorial"
#-----------------------------------------------
SET endbutton:"ok"
WHILE (endbutton = "ok") {
  CC "AdoScript" EDITBOX text: (adoscript)
    title:"Enter the code you want to test..."
    oktext:"Run" fontname:"Arial" fontheight: 14
  # --> RESULT endbutton:strValue text:strValue
  IF (endbutton = "ok") {
    SETG adoscript:(text)
    EXECUTE (text)
  }
}
#-----------------------------------------------
```

Hint: As you develop more and more algorithms for your modeling method, the contents of the library attribute “External coupling” will increase in size. In order to maintain a good overview of your code, it is advisable to insert so-called “separator lines” before and after each menu item definition, context menu item definition or event handler definition as seen in the example above. **The hash character (#) is used for commenting an entire line**.

6. Click “Apply”



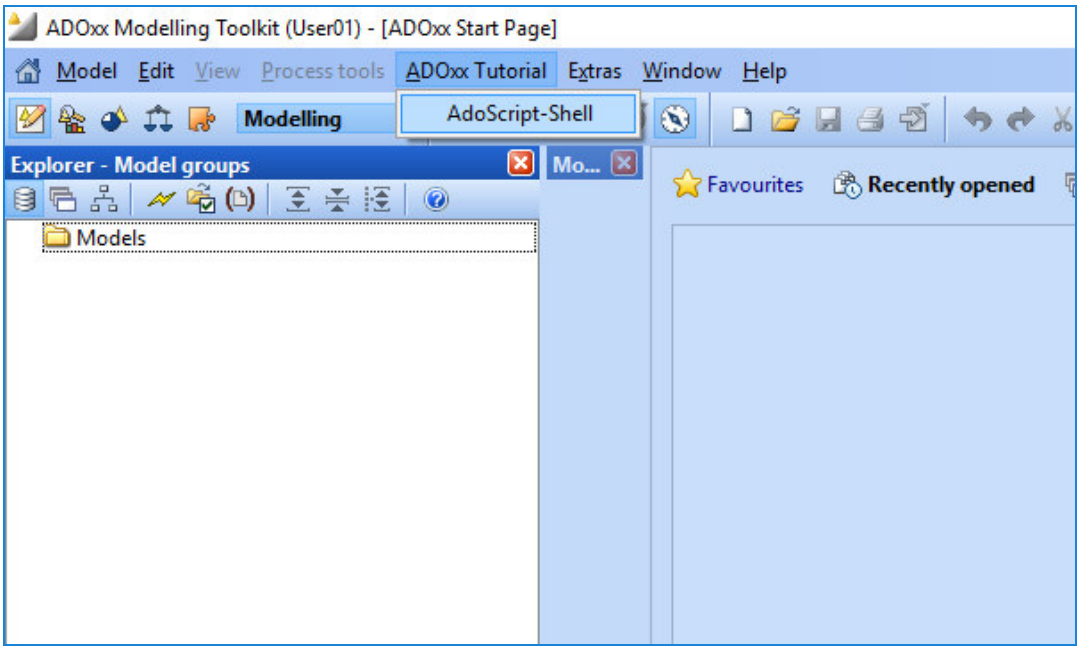
7. Click “Apply” in the “Library attributes” window



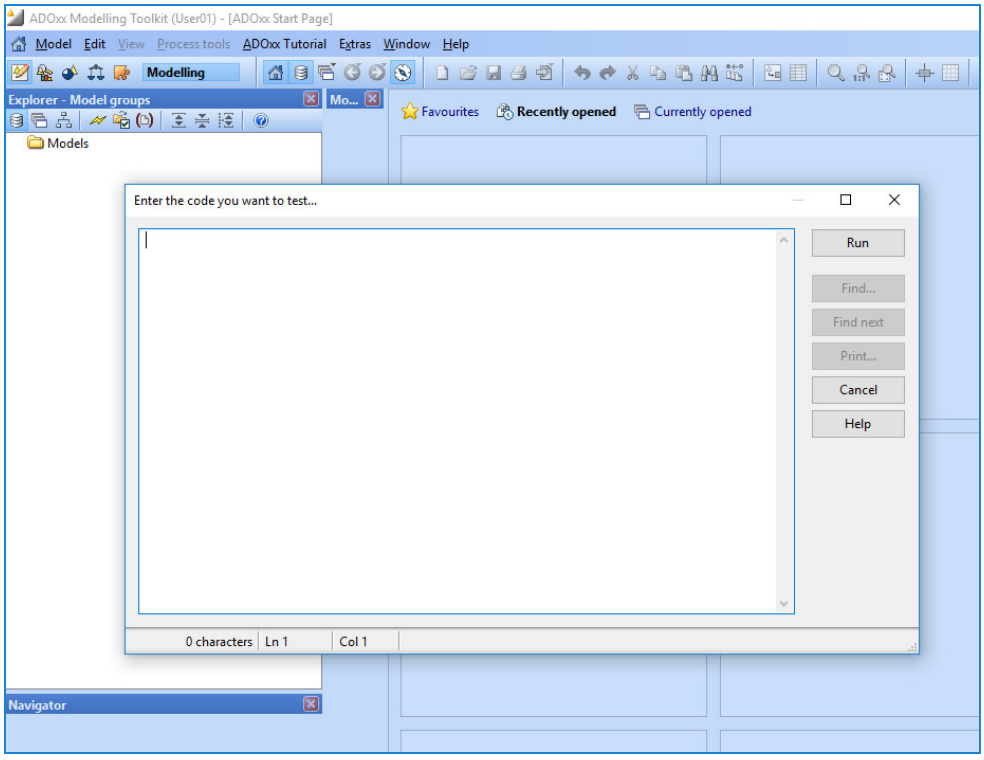
8. Click “Yes” to save changes.

2.2. Test the newly created menu item

1. Open the ADOxx Modelling Toolkit. If the ADOxx Modelling Toolkit is already open, you will have to close it and open it again, for the newly defined menu item to be loaded.
2. Log in with the username and password created when implementing the modeling language.
3. Click the menu item “AdoScript-Shell” located under the menu “ADOxx Tutorial”



4. The AdoScript Debug Shell window will open



5. Click “Cancel” to close the window.

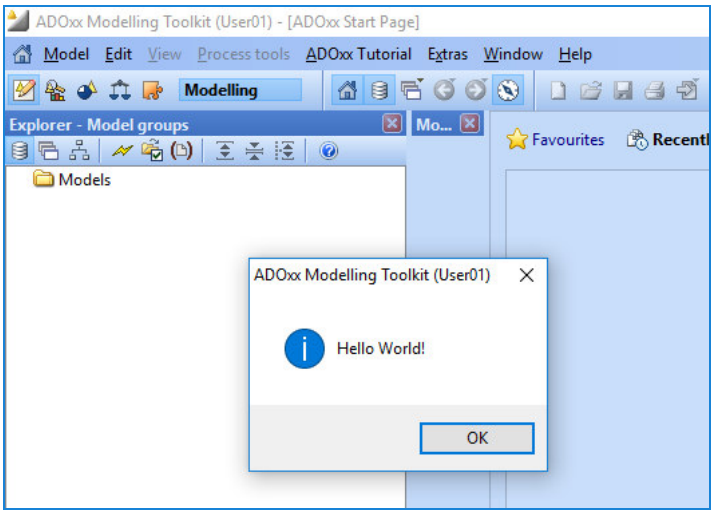
2.3. Execute 'Hello World!" by means of AdoScript Shell

2.3.1. "Hello World!" – basic

1. Open the AdoScript Shell as described at 2.2.
2. Enter the following command:

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

3. A message box will appear:



4. Close the window by clicking "OK"

2.3.2. "Hello World!" – with variable

5. Open the AdoScript Shell
6. Enter the following code and click „Run":

```
1  SET sText: ("Hello World")
2  CC "AdoScript" INFOBOX (sText)
```

7. A message box with the text "Hello World!" will appear
8. Close the window by clicking "OK"

2.4. Execute 'Hello World' by means of a Separate Menu Button

Note: If you are familiar with the execution of AdoScript files triggered by a menu button, then you can download the library, which already includes this functionality as well as the fuctionalities implemented in previous sections and continue with section **3.**

You can find the library in the Method Engineering Tutorial Package under "*<mypath>\Method-Engineering-Package\Tutorial_Libraries\Mechanisms and Algorithms Implementation Libraries*"

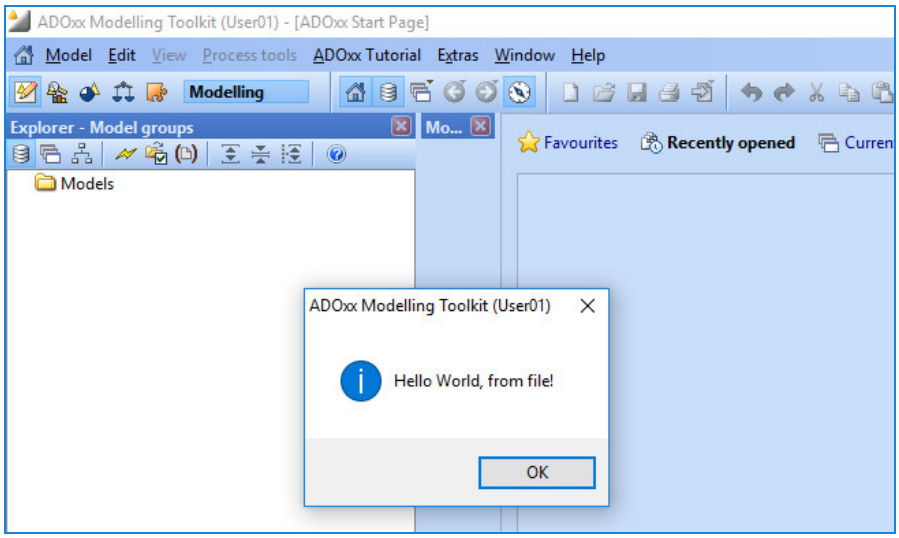
1. Open-up a new file in a text editor and enter the following text:

```
1  SET sText: ("Hello World, from file!")
2  CC "AdoScript" INFOBOX (sText)
```

2. Save the file in a folder of your choice, e.g. “C:\temp\ADOxx-Training” using the name “adoscript1.asc”
3. Create another menu button “*Hello World*” in an entirely new menu “*ADOxx Training*” as described in subchapter 2.1.:

```
1  #----------------------------------------------
2  ITEM "Hello World"
3   acquisition:"ADOxx Tutorial" modeling:"ADOxx Tutorial"
4   analysis:"ADOxx Tutorial" simulation:"ADOxx Tutorial"
5   evaluation:"ADOxxTutorial" importexport:"ADOxx Tutorial"
6  #----------------------------------------------
7  EXECUTE file: ("C:\\temp\\ADOxx-Training\\adoscript1.asc")
8  #----------------------------------------------
```

4. By clicking on the new button “*Hello World*” the file “adoscript1.asc” will be executed:



3. Returning the Attribute Values of Objects

After having successfully added new menu point *“AdoScript-Shell”* in the menu “*ADOxx Tutorial”,* in this chapter, we will implement a few AdoScript samples for returning different attribute values of Objects present in models that are open.

A more detailed description of the message port commands used in these examples, as well as further message port commands, can be found in the ADOxx Development Toolkit help; you can access the help by pressing the F1 key.

3.1. Ascertaining the Active Model

The message port command used for retrieving the active model is:

GET_ACT_MODEL

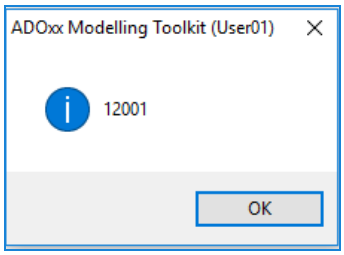
The command is part of the message port “Modeling”, it takes no parameters and returns the id of the active model in the global variable “modelid”.

1. Open the AdoScript Shell
2. Enter the following code and click „Run“:

Note: In order to get a model id you need to open a model

```
1 CC "Modeling" GET_ACT_MODEL
2 # --> RESULT modelid: intValue
3 CC "AdoScript" INFOBOX (STR modelid)
```

3. A message box with the id of the current model will appear



4. Close the window by clicking “OK”

3.2. The debug mode

As you have noticed, the values of the output parameters of each command call are not implicitly displayed in the ADOxx Modelling Toolkit. In the example at 3.1 we have called the message port command “INFOBOX” for displaying the value returned by the command “GET_ACT_MODEL”.

In order to avoid a substantial increase in size of the code by adding an “INFOBOX” command call after each message port command call, every command call can be made in debug mode. This can be achieved by adding the “debug” keyword between the name of the message port and the name of the command. When a command call (CC) is made in debug mode, a message box is displayed after executing the command. This message box contains all the input parameters and all the output variables of the command, in pairs of name and value.

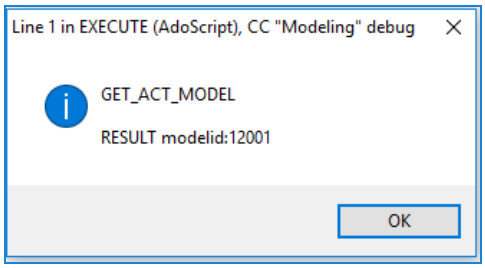
The debug mode is particularly helpful when you work with large AdoScript files or command sequences.

1. Open the AdoScript Shell

2. Enter the following code and click „Run“:

```
1  CC "Modeling" debug GET_ACT_MODEL
```

3. A message box containing the id of the current model will appear



4. Close the window by clicking “OK”

Hint: Beside input and output variable names and values, the message box also contains information about the line in the AdoScript text where the call has been made.

3.3. Ascertaining all Objects of Class “A”

After having determined the id of the current model, the next step is retrieving all objects of the desired class. For this, the following message port command will be used:

GET_ALL_OBJS_OF_CLASSNAME

1. Open the AdoScript Shell
2. Enter the following code:

```
1  CC "Modeling" debug GET_ACT_MODEL
2  # --> RESULT modelid: intValue
3  SET nActModelid: (modelid)
```

3. Open the Help in the ADOxx Development Toolkit (by pressing the F1 key)
4. Click the “Index” tab and type in “GET_ALL_OBJS_OF_CLASSNAME” until the item “GET_ALL_OBJS_OF_CLASSNAME” is displayed in the list to the left (Figure 1 at the end of this subchapter)
5. Click on the item “GET_ALL_OBJS_OF_CLASSNAME” in the list to the left and in the right section of the help window the detailed list of all commands for the message port corresponding to the selected command will be displayed (in this case, “Core”)
6. Click in the right section of the help window and search for the detailed description of the required command; use the key combination Ctrl + F for automatic search.
7. Select and copy the format of the command
8. Paste the copied command after the last line of code in the AdoScript Shell:

GET_ALL_OBJS_OF_CLASSNAME modelid: *id* classname: *strValue*
--> RESULT ecode: *intValue* objids: *list* .

9. Edit the first line added by adding the “CC” keyword, the message port name “Core” and the “debug” keyword before the command name

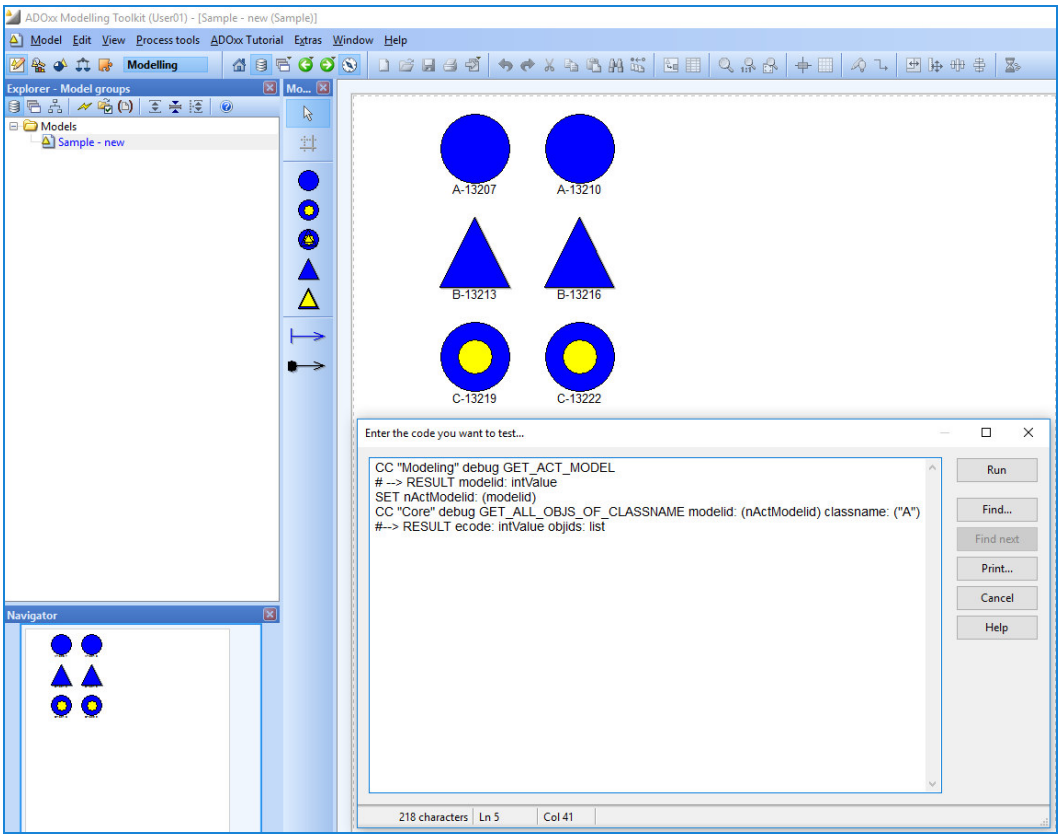
10. Give the proper values to the input parameters of the command:

```
4  CC "Core" debug GET_ALL_OBJS_OF_CLASSNAME modelid: (nActModelid) classname: ("A")
```

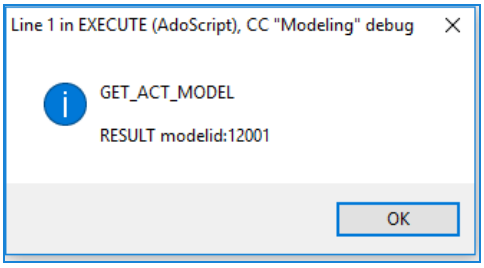
11. Comment the second line containing the result variables using the character “#” :

```
5  #--> RESULT ecode: intValue objids: list
```

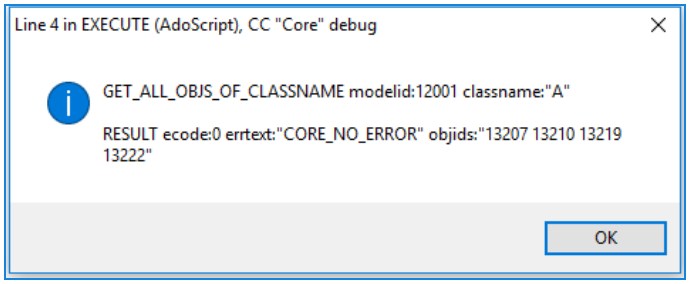
12. Click “Run” to run the script



13. A message box containing the id of the current model will appear as a result of using the “debug” keyword on the command call “GET_ACT_MODEL”



14. After closing this message box, another message box will be displayed, containing the parameters and output of the command call “GET_ALL_OBJS_OF_CLASSNAME”



15. Close the window by clicking “OK”

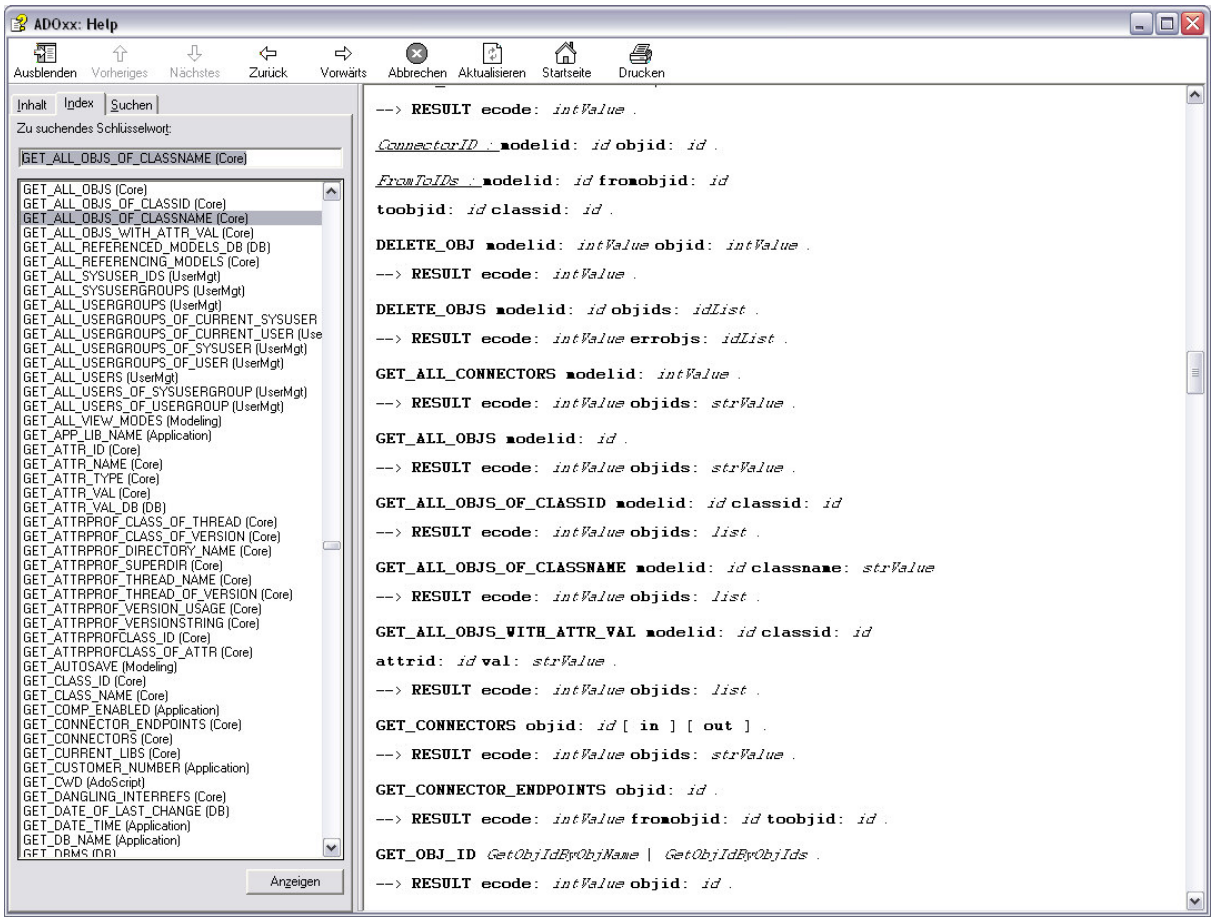


Figure 1 Screenshot of the ADOxx help window

Note: In the example above, not only the IDs of the two objects that are instances of class “A” are returned, but also the IDs of the instances of class “C”, as class “C” has been derived from class “A” (see the “ADOxx 1.5 Cookbook for Implementing Modelling Language”). The returned result is a string variable containing the list of IDs separated by blanks (“ “). The IDs returned in the string variable can be accessed using the **FOR … in: …** statement as described in the next subchapter.

Hint: It is recommended to keep in your AdoScript code all the information about the input and output variables of a message port command (see steps 8. – 11.). This helps keeping track of the names and types of the variables in your code. It is even more helpful if you are storing your code in files that are executed by clicking a menu item as described in subchapter 2.4. and you need to review your code from time to time.

Hint: Even if you are writing AdoScript code for event handlers or menu items in the "External coupling" library attribute, for ease of testing it is recommended to write your code in an external file and read the AdoScript code from that file every time the code needs to be run. This saves you time with testing, since after every change in the library attributes you have to restart the ADOxx Modelling Toolkit for the changes to take effect. Using an open-use text editor like "Notepad++" is recommended and syntax highlighters for this text editor are available on the www.adoxx.org webpage: ADOxx Development Languages: Syntax Support.

3.4. Ascertaining the Names of all Objects of the Class 'A'

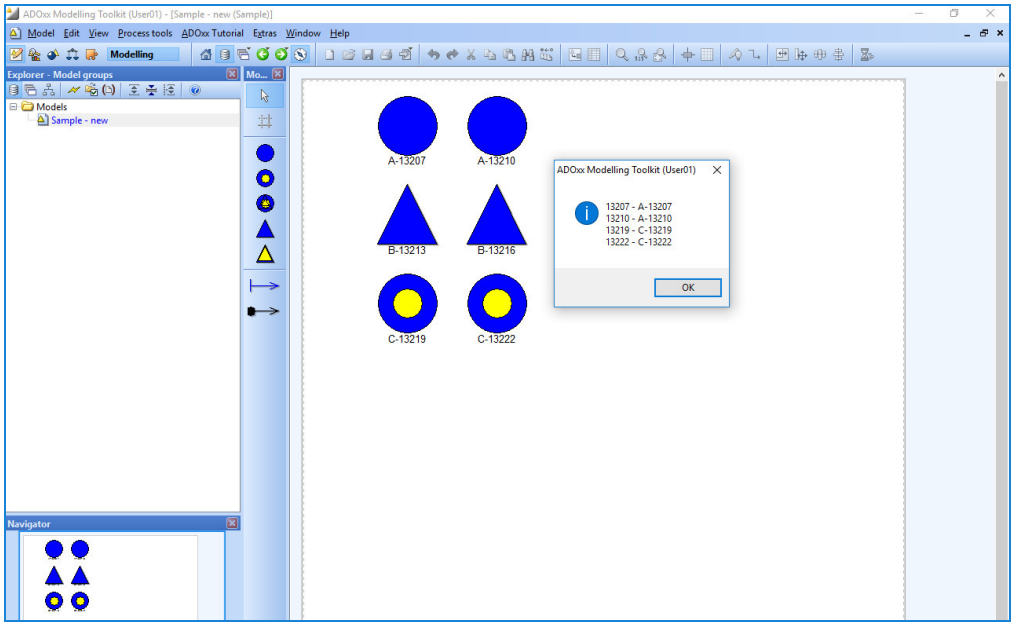
In this exercise all names of all instances of class "A" of the active model should be identified, stored in form of *string variables* and displayed. This can be realised by means of a **FOR** loop (string-token-form). The following commands will be used:

GET_ACT_MODEL
GET_ALL_OBJS_OF_CLASSNAME
GET_CLASS_ID
GET_OBJ_NAME

1. Open the AdoScript Shell
2. Enter the following code and click "Run":

```
1  CC "Modeling" GET_ACT_MODEL
2  # --> RESULT modelid: intValue
3  SET nActModelid: (modelid)
4  CC "Core" GET_ALL_OBJS_OF_CLASSNAME modelid: (nActModelid) classname: "A"
5  #--> RESULT ecode: intValue objids: list .
6
7  SET sObjIds: (objids)
8  CC "Core" GET_CLASS_ID classname: "A" bp-library
9  # --> RESULT ecode: intValue classid: intValue .
10
11 SET nAClassId: (classid)
12 SET sResult: ("")
13 FOR i in: (sObjIds) {
14   SET nCrtObjId: (VAL i)
15   CC "Core" GET_OBJ_NAME objid: (nCrtObjId)
16   # --> RESULT ecode: intValue objname: strValue .
17   SET sResult: (sResult + STR nCrtObjId + " - " + objname +"\n")
18 }
19 CC "AdoScript" INFOBOX (sResult)
```

3. A message box containing a list with pairs id – name will be displayed:



4. Click “OK” to close the message box.

Note: This exercise is also an example for the usage of a **FOR** loop and of the keyword **VAL** for converting a string into an integer. The conversion from number variables (integer or float) into strings is done using the **STR** keyword and will be exemplified later in this document.

3.5. Displaying all attributes of all Objects of the Class ‘A’

In this subchapter, the values of all visible attributes of all objects of class “A” in the active model will be identified, stored in form of a large *string variable* and displayed. The following message port commands will be used (in addition to the commands used in subchapter 3.4.):

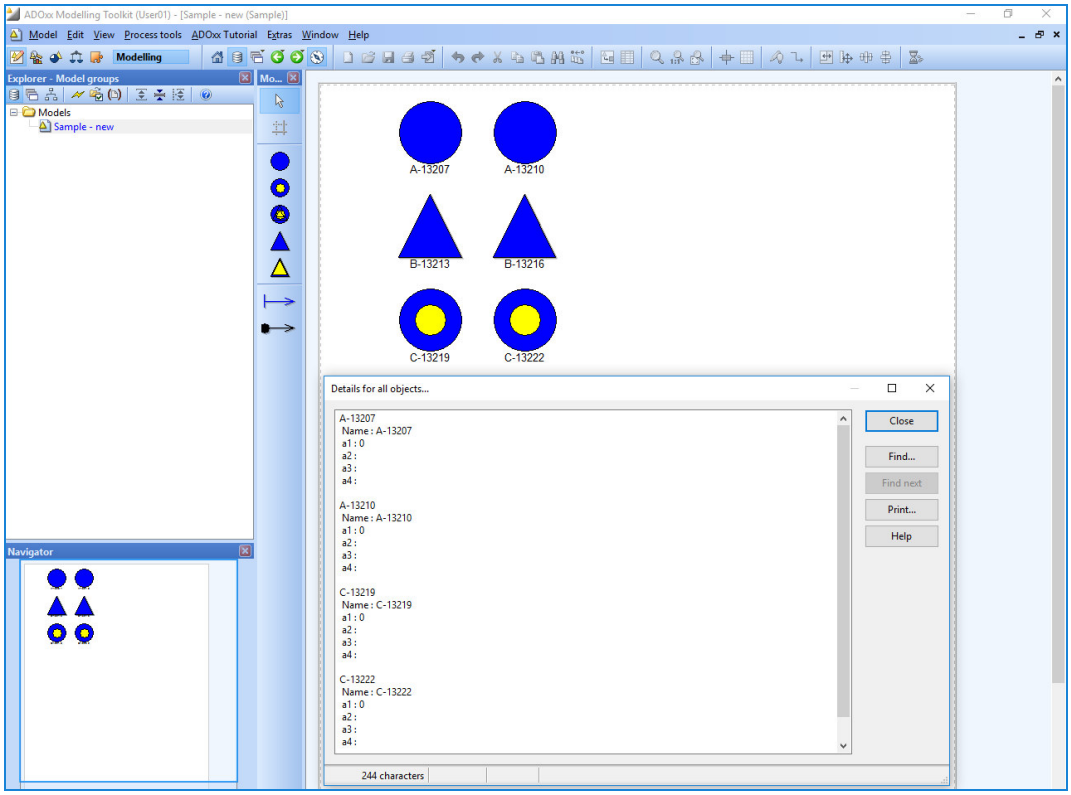
GET_ALL_NB_ATTRS
GET_ATTR_NAME
GET_ATTR_VAL
VIEWBOX

1. Open the AdoScript Shell
2. Enter the following code and click “Run”:

```
1  CC "Modeling" GET_ACT_MODEL
2  # --> RESULT modelid: intValue
3  SET nActModelid: (modelid)
4  CC "Core" GET_ALL_OBJS_OF_CLASSNAME modelid: (nActModelid) classname: "A"
5  #--> RESULT ecode: intValue objids: list .
6
7  SET sObjIds: (objids)
8  CC "Core" GET_CLASS_ID classname: "A" bp-library
9  # --> RESULT ecode: intValue classid: intValue .
```

```
10
11  SET nAClassId: (classid)
12  CC "Core" GET_ALL_NB_ATTRS classid: (nAClassId)
13  #--> RESULT ecode: intValue attrids: strValue .
14
15  SET sAttrIdsA: (attrids)
16  SET sResult: ""
17  FOR i in: (sObjIds) {
18    SET nCrtObjId: (VAL i)
19    CC "Core" GET_OBJ_NAME objid: (nCrtObjId)
20    # --> RESULT ecode: intValue objname: strValue .
21
22    SET sResult: (sResult + objname + "\n")
23    FOR j in: (sAttrIdsA) {
24      SET nAttrId: (VAL j)
25      CC "Core" GET_ATTR_NAME attrid: (nAttrId)
26      #--> RESULT ecode: intValue attrname: strValue .
27
28      CC "Core" GET_ATTR_VAL objid: (nCrtObjId) attrid: (nAttrId) as-string
29      #--> RESULT ecode: intValue val: anyValue .
30
31      SET sResult: (sResult + " " + attrname + " : " + val + "\n")
32    }
33    SET sResult: (sResult + "\n")
34  }
35  CC "AdoScript" VIEWBOX text: (sResult) title: "Details for all objects..."
```

3. A message box containing a list with pairs id – name for each instance of class “A” will be displayed:



4. Click “Close” to close the message box.

3.6. Constraints on Attribute Values

Return only the Attribute Values that are nonempty, using the interrogation:

IF

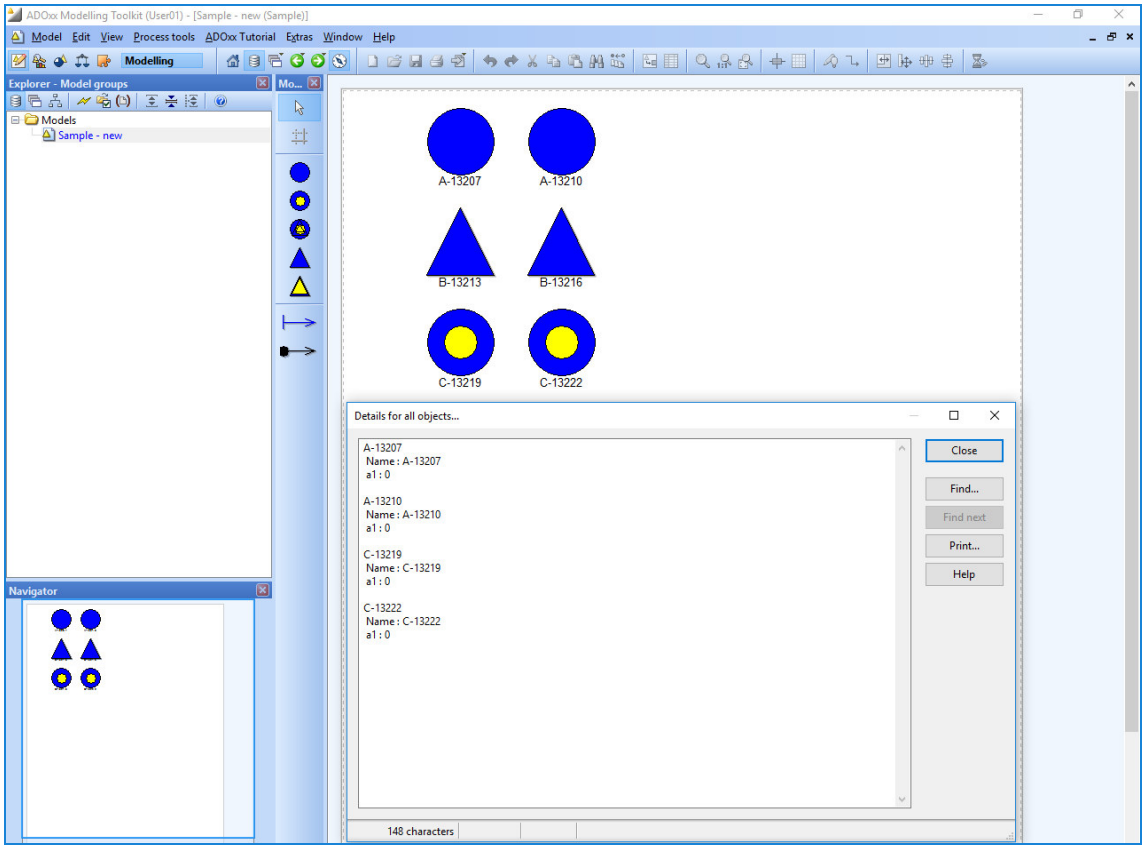
1. Open the AdoScript Shell
2. Enter the following code and click “Run”:

Note: Only the red marked lines differ from the code in section 3.5

```
1  CC "Modeling" GET_ACT_MODEL
2  # --> RESULT modelid: intValue
3  SET nActModelid: (modelid)
4  CC "Core" GET_ALL_OBJS_OF_CLASSNAME modelid: (nActModelid) classname: "A"
5  #--> RESULT ecode: intValue objids: list .
6
7  SET sObjIds: (objids)
8  CC "Core" GET_CLASS_ID classname: "A" bp-library
9  # --> RESULT ecode: intValue classid: intValue .
10
11 SET nAClassId: (classid)
12 CC "Core" GET_ALL_NB_ATTRS classid: (nAClassId)
13 #--> RESULT ecode: intValue attrids: strValue .
14
```

```
15  SET sAttrIdsA: (attrids)
16  SET sResult: ""
17  FOR i in: (sObjIds) {
18   SET nCrtObjId: (VAL i)
19   CC "Core" GET_OBJ_NAME objid: (nCrtObjId)
20   # --> RESULT ecode: intValue objname: strValue .
21
22   SET sResult: (sResult + objname + "\n")
23   FOR j in: (sAttrIdsA) {
24    SET nAttrId: (VAL j)
25    CC "Core" GET_ATTR_NAME attrid: (nAttrId)
26    #--> RESULT ecode: intValue attrname: strValue .
27
28    CC "Core" GET_ATTR_VAL objid: (nCrtObjId) attrid: (nAttrId) as-string
29    #--> RESULT ecode: intValue val: anyValue .
30    IF (val != "") {
31     SET sResult: (sResult + " " + attrname + " : " + val + "\n")
32    }
33   }
34   SET sResult: (sResult + "\n")
35  }
36  CC "AdoScript" VIEWBOX text: (sResult) title: "Details for all objects..."
```

3. A message box containing a list with pairs id – name for each instance of class “A” will be displayed:



4. Click “Close” to close the message box.

4. Menu items and event handlers

4.1. Implement a Menu Item for changing attribute values

Note: If you know how to change the attribute values of objects using a menu button that triggers an AdoScript file, then you can download the library, which already includes this functionality as well as the fuctionalities implemented in previous sections and continue with section **4.2.**

You can find the library in the Method Engineering Tutorial Package under “*<mypath>\Method-Engineering-Package\Tutorial_Libraries\Mechanisms and Algorithms Implementation Libraries*”

In this subchapter you will add a new menu item that when clicked changes the attribute values of attribute “a1” for every instance of class “A” by adding 1 to the original value. The following message port command will be used:

SET_ATTR_VAL

1. Open a text editor and create a new file.
2. Save the empty file in the folder “C:\temp\ADOxx-Training” using the name “item_4_1.asc”
3. Create another menu button “Change a1 Values” in the menu “ADOxx Training”:

```
#----------------------------------------------
ITEM "Change a1 Values"
 acquisition:"ADOxx Tutorial" modeling:"ADOxx Tutorial"
 analysis:"ADOxx Tutorial" simulation:"ADOxx Tutorial"
 evaluation:"ADOxxTutorial" importexport:"ADOxx Tutorial"
#----------------------------------------------
EXECUTE file: ("C:\\temp\\ADOxx-Training\\item_4_1.asc")
#----------------------------------------------
```

4. Apply all changes and save the application library.
5. Open the file “item_4_1.asc”
6. Type in the following code :

```
CC "Modeling" GET_ACT_MODEL
# --> RESULT modelid: intValue
IF (modelid = -1) {
  CC "AdoScript" INFOBOX "No model is open, so no changes can be applied"
}
ELSE {
  SET nActModelId: (modelid)
  CC "Core" GET_ALL_OBJS_OF_CLASSNAME modelid: (nActModelId) classname: "A"
  #--> RESULT ecode: intValue objids: list .
  SET sObjids: (objids)
  CC "Core" GET_CLASS_ID classname: "A" bp-library
  # --> RESULT ecode: intValue classid: intValue .
```

```
13   SET nAClassId: (classid)
14   FOR i in: (sObjids) {
15     SET nCrtObjId: (VAL i)
16     CC "Core" GET_ATTR_VAL objid: (nCrtObjId) attrname: "a1"
17     # --> RESULT ecode: intValue val: anyValue
18     SET nCrtValue: (val + 1)
19     CC "Core" SET_ATTR_VAL objid: (nCrtObjId) attrname: "a1" val: (nCrtValue)
20     # --> RESULT ecode: intValue .
21    }
22   CC "AdoScript" INFOBOX "Attribute values have been changed for all objects!"
23  }
```

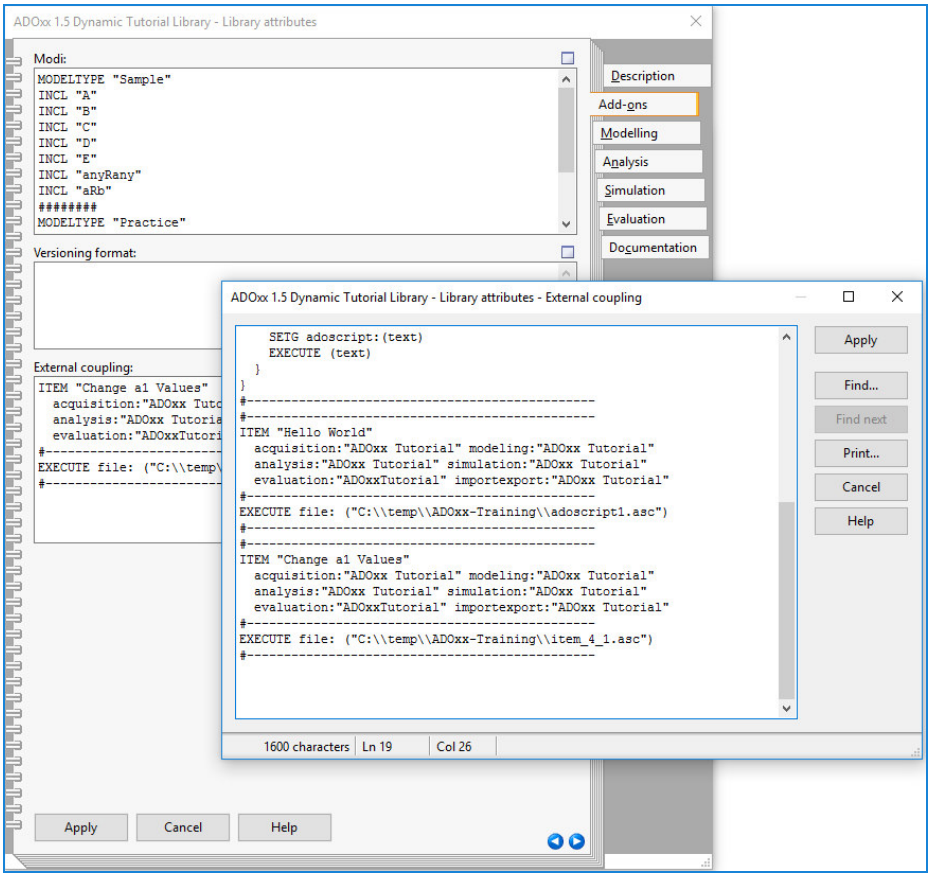
7. Save the AdoScript file “item_4_1.asc”
8. Open the ADOxx Modelling Toolkit and test the new menu item.

4.2. Implement an event handler “AfterCreateModelingConnector”

In this subchapter you will add an event handler to the “External coupling” library attribute for the event “AfterCreateModelingConnector”. When the event is triggered, an INFOBOX will be shown, displaying the following information: name and class of the From-Object, name and class of the To-Object and name of the Connector.

ON_EVENT “**AfterCreateModelingConnector**” { … }
toobjid
fromobjid
objid
PROCEDURE

1. Open up the “Library Management” section
2. Click the “Library attributes” button
3. Select the “Add-ons” tab
4. Open the “External coupling” library attribute



5. Add the following code to implement the event handler for the event "AfterCreateModellingConnector" :

```
#--------------------------------------------------------
ON_EVENT "AfterCreateModelingConnector"
{
 #modelid - the model ID of the modified model
 #objid - the ID of the newly inserted instance
 #classid - the ID of this instance's class
 #fromobjid - the ID of the from-object of the new connector
 #toobjid - the ID of the new connector's target object
 #origin one of these numeric values: 0 - modelled by the user, 1 - pasted, 2 - via "undo".

 SET nModelid: (modelid)
 SET nObjId: (objid)
 SET nClassId: (classid)
 SET nFromObjId: (fromobjid)
 SET nToObjId: (toobjid)
 SET nOrigin: (origin)

 SET sFromObjInfo: ""
 GET_OBJECT_INFO (nFromObjId) result: sFromObjInfo
 SET sToObjInfo: ""
 GET_OBJECT_INFO (nToObjId) result: sToObjInfo
 SET sResult: ("From " + sFromObjInfo + "To " + sToObjInfo)
```

```
23   CC "Core" GET_CLASS_NAME classid: (nClassId)
24   #--> RESULT ecode: intValue classname: strValue isrel: intValue
25   SET sResult: ( "Relation " + classname + "\n"+ sResult)
26   CC "AdoScript" INFOBOX (sResult)
27
28   PROCEDURE GET_OBJECT_INFO integer: nObjectId result: reference
29   {
30    CC "Core" GET_CLASS_ID objid: (nObjectId)
31    #--> RESULT ecode: intValue classid: intValue isrel: intValue .
32    CC "Core" GET_CLASS_NAME classid: (classid)
33    #--> RESULT ecode: intValue classname: strValue isrel: intValue .
34    CC "Core" GET_OBJ_NAME objid: (nObjectId)
35    # --> RESULT ecode: intValue objname: strValue .
36    SET result: ("object " + objname + " of class " + classname + " \n")
37   }
38  }
39  #-------------------------------------------------------
```

6. Apply all changes and save the application library
7. Re-open the ADOxx Modelling Toolkit and test the event handler by connecting two objects.

Hint: The example above also shows the usage of procedures for simplifying repetitive code.

Hint: It is recommended to save the global variables that are passed as parameters when the event is triggered (in our example: modelid, objid, classid, fromobjid, toobjid, origin) so they are not lost when executing other command calls in the event handler.

Hint: It is also recommended to describe the passed parameters at the beginning of the event handler so you can have a better overview of more complex event handlers. The description of the events that are triggered within the ADOxx Modelling Toolkit and the parameters passed can be found in the ADOxx help. You can access the ADOxx help by pressing the F1 key and then type “event handler” in the index.

Hint: The code for the event handler can be implemented in an external file, in the same manner as described at 4.1. The only difference is that the parameters passed to the event handler should be added before the text read from the .asc file. For the example above, the solution would be like this:

```
1  #-------------------------------------------------------------------------
2  ON_EVENT "AfterCreateModelingConnector"
3  {
4   #modelid - the model ID of the modified model
5   #objid - the ID of the newly inserted instance
6   #classid - the ID of this instance’s class
7   #fromobjid - the ID of the from-object of the new connector
8   #toobjid - the ID of the new connector’s target object
9   #origin one of these numeric values: 0 - modelled by the user, 1 - pasted, 2 - via "undo".
```

```
10
11   SET sParams: ("")
12   SET sParams: (sParams+ "SET nModelid: (" + ( STR modelid) + ") \n")
13   SET sParams: (sParams+ "SET nObjId: (" + ( STR objid) + ") \n")
14   SET sParams: (sParams+ "SET nClassId: (" + ( STR classid) + ") \n")
15   SET sParams: (sParams+ "SET nFromObjId: (" + ( STR fromobjid) + ") \n")
16   SET sParams: (sParams+ "SET nToObjId: (" + ( STR toobjid) + ") \n")
17   SET sParams: (sParams+ "SET nOrigin: (" + ( STR origin) + ") \n")
18   CC "AdoScript" FREAD file: ("C:\\temp\\ADOxx-Training\\handler_4_2.asc")
19   #--> RESULT text:strValue ecode:intValue
20   EXECUTE (sParams + text)
21  }
22  #---------------------------------------------------------------------
```

where "C:\\temp\\ADOxx-Training\\handler_4_2.asc" is the path and name of the file where the implementation code is stored.

Content of the handler_4_2.asc file:

```
1   SET sFromObjInfo: ""
2   GET_OBJECT_INFO (nFromObjId) result: sFromObjInfo
3   SET sToObjInfo: ""
4   GET_OBJECT_INFO (nToObjId) result: sToObjInfo
5   SET sResult: ("From " + sFromObjInfo + "To " + sToObjInfo)
6   CC "Core" GET_CLASS_NAME classid: (nClassId)
7   #--> RESULT ecode: intValue classname: strValue isrel: intValue
8   SET sResult: ( "Relation " + classname + "\n"+ sResult)
9   CC "AdoScript" INFOBOX (sResult)
10
11  PROCEDURE GET_OBJECT_INFO integer: nObjectId result: reference
12  {
13   CC "Core" GET_CLASS_ID objid: (nObjectId)
14   #--> RESULT ecode: intValue classid: intValue isrel: intValue .
15   CC "Core" GET_CLASS_NAME classid: (classid)
16   #--> RESULT ecode: intValue classname: strValue isrel: intValue .
17   CC "Core" GET_OBJ_NAME objid: (nObjectId)
18   # --> RESULT ecode: intValue objname: strValue .
19   SET result: ("object " + objname + " of class " + classname + " \n")
20  }
```