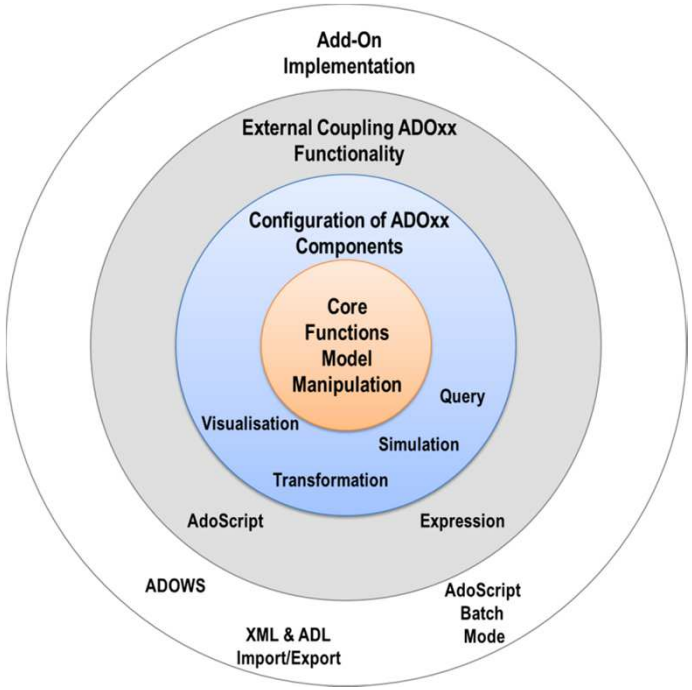


3. EXTERNAL COUPLING ADOXX FUNCTIONALITY

What is AdoScript?

AdoScript is the macro language of ADOxx. It is based on LEO and is build procedural. Through AdoScript the user has access to a huge number of ADOxx functionalities.

AdoScript is a mighty tool which allows huge extension possibilities with low programming effort.

Examples:

- New menu entries
- Integration of new tools
- Realisation of specific model checking
- Realisation of new interfaces
- Additional add-on-programming

How is AdoScript used?

AdoScript can be executed on different ways. So it can be used where it is needed:

As menu entry: For manual execution
(e.g. transformation procedures, evaluation scenarios)

In events: If specific actions are executed, an AdoScript can be automatically called.
(e.g. a special dialogue replaces the standard dialogue window)

Notebook via Programmcall

Automatic over Command prompt

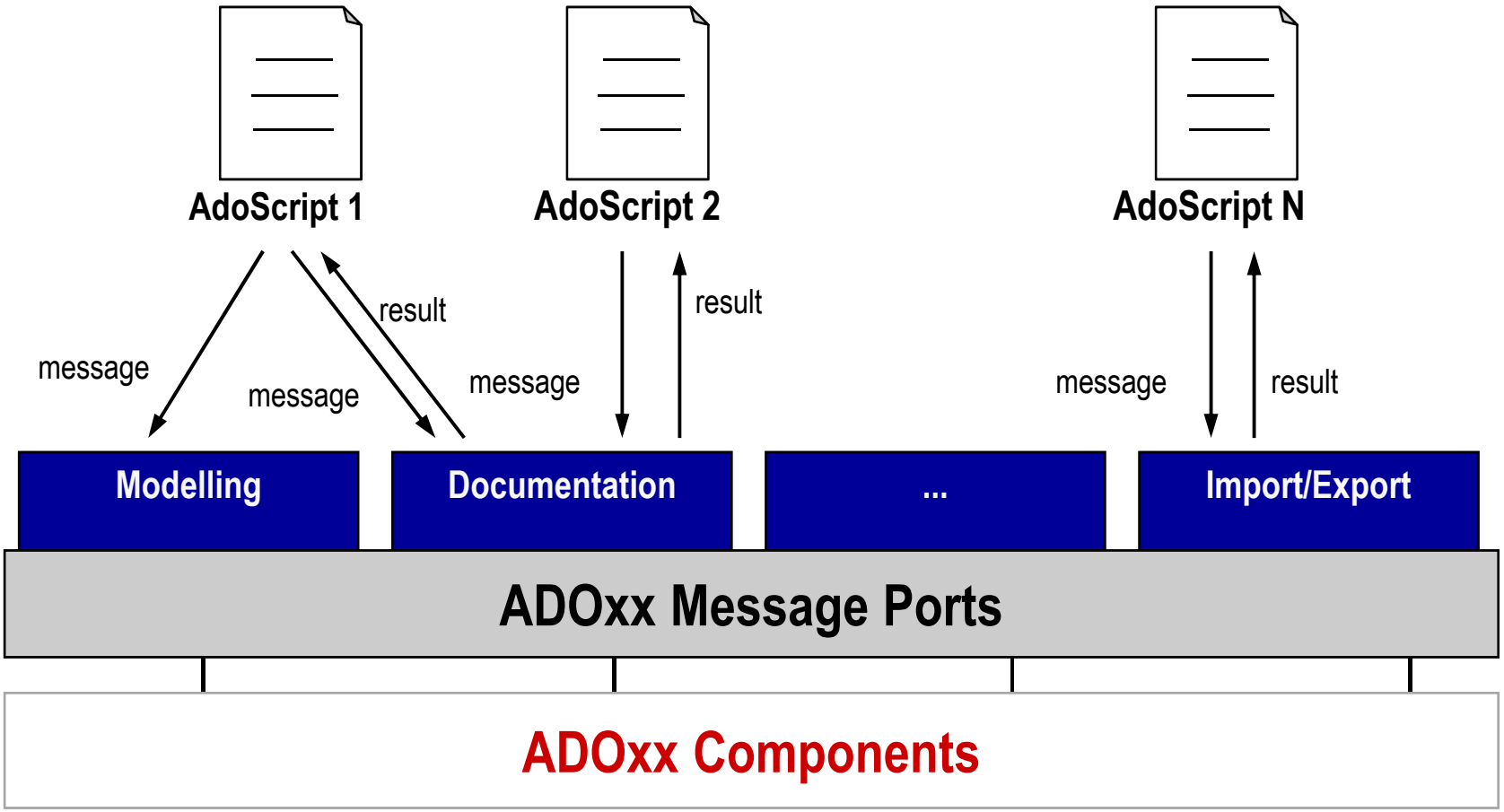
```
ECHO CC "AdoScript" FREAD file:("batchupd.adoscript")
EXECUTE (text) CC "Application" EXIT |
areena -ubatchupd -pbatchupd -dADOxxdb -ssqlserver -e
```

From AdoScript-Shell

Integration of AdoScript:

The Message Port-Concept

AdoScript can be integrated with „External binding“ or „Programm call“.



Programmable through scripting APIs

- **Method-specific development of functionalities through scripting**
- Function calls/APIs of the platform (realized in C++) are possible through scripting language AdoScript.
- Categorisation of APIs called „Messageport“.

Component APIs

Messageport **Acquisition**
Messageport **Modeling**
Messageport **Analysis**
Messageport **Simulation**
Messageport **Evaluation**
Messageport **ImportExport**
Messageport **Documentation**
Messageport **AQL**

UI APIs

Messageport **AdoScript**
Messageport **CoreUI**
Messageport **Explorer**

Manipulation APIs

Messageport **Core**
Messageport **DB**
Messageport **UsrMgt**

Application APIs

Messageport **Drawing**
Messageport **Application**

About 400 APIs are available.

Documentation of MessagePorts and AdoScript Call Signatures

Useful Hint

Every Command Call stores the result in global variables

=> HINT:

Store right after the CC required global variables in local variable to avoid overwriting by the next CC

Tracking the global variables with “debug”

=> HINT:

Use the keyword debug during CC “CC “xxx” debug” to track the status of variables

Variables are allocated with values, distinguish if you manipulate the variable v1, or the value of the variable (v1)

=> HINT:

- use VAL and STR to convert strings to integer and vice versa
- use tokcnt to count tokens in a result list
- use () to get the value of a variable
- use CM to convert into centimetre

Data Type Conversion

- **STR** *val* Converts a *value* into a string.
-
- **VAL** *str* Parses the string and returns that value.
- **CM** *realVal* Converts a real value in centimetres into a centimetre
- **PT** *realVal* Converts a real value in points into a measure value.
- **uistr (***val***,** *digits***)** Converts a real value in a string
- **uival (***str***)** Converts a string value in a real value
- **CHR** *intVal* Returns the character of for the character code provided in *intVal*. Return type is *str*. For example: **CHR 65** = **"A"**.
- **ASC** *str* Returns the character code for the character passed in *str*. For example: **ASC "A" = 65**.
- **INT** *realVal* Returns the an *intVal*. The *realVal* is converted to integer by truncating digits after the decimal point.