Alephino 5.0 Scripting language

From: Swami Wiki **Date: September 19th 2014**

Contents

General

Alephino Scripts are an instrument to check or manipulate data records. Error messages can be produced, data fields can be inserted, updated or deleted.
A script contains a sequence of statements in a ASCII text file (script file). A script file can contain several scripts.

There are following scripts in the Alephino standard version:

- A script **CHECK** to check data records in the script file **mabscript.txt** resp. **marcscript.txt**.
- A script **COMPL** to complete data records in the script file **mabscript.txt** resp. **marcscript.txt**.
- Scripts **CHECKxxx** to check data fields in the script file **mabfield.txt** resp. **marcfield.txt**. **xxx** means the concerned field tag.

Those scripts have following parameters:

- Parameter P1: File name (library, f.e. M-TIT).
- Parameter P2: User name.
- Parameter P3: System number of the data record.
- Parameter P4: Content of the data field (only scripts for field checks).

Objects

Following objects can be manipulated in a script:

- String constants
- Sets of string constants
- Numeric values (integers)
- Content of data fields
- Variables
- Parameter

String constants

Strings are sequences of characters included in double quotation marks.

Example:

```
"This is a text"
```

Sets of string constants

A set of string constants are strings separated by comma and included in curly brackets.

Example:

```
{ "Text1" , "Text2" , "Text3" }
```

Numeric values

Only integer values are allowed.

Content of data fields

The content of a data field can be addressed as follows:

```
:field-tag (Ex.: :100)
```

The content of a data field with special indicators:

```
:field-tag/"indicators" (Ex.: :100/"a")
```

The content of repeatable data fields:

```
:field-tag.index resp. :field-tag/"indicator".index (Ex.: :100.2)
```

Subfields can be addressed by:

```
field-tag$subfield-tag
```

The above mentioned combinations are allowed for field-tag. If there are repeatable subfields, the subfield-tag can be qualified by .index.

So the most complex address for data of a subfield is:

```
:field-tag/"indicator".index$subfield-tag.index
```

Example:

```
:100/"i".2$u.1 is the data of the first subfield u of the second field 100 with indicator i.
```

Variables

Following variables can be defined:

Variables of strings by the identifier **STRING**. Variables of string sets by the identifier **STRSET**. Variables of integers by the identifier **INT**.

Ex. for a string variable definition:

```
STRING var
```

Values can be assigned to variables. Ex.:

```
var = "Content of variable"
```

It is also possible to initialize a variable at definition time.

Ex.:

```
STRING var = "Content of variable"
```

Parameter

Parameters are addressed by **&P** followed by the number of the parameter, i.e. **&P1** is the first, **&P2** is the second parameter etc.

Statements

The statements of a script are enclosed in a beginning

PROC scriptname

and an ending

END PROC

There are following types of statements:

- Assignment of objects to data fields or variables
- Set (error) message
- Call of subordinate scripts (procedures)
- Control statements (IF, WHILE, LOOP, CHOOSE)

Additionally there are several built-in functions which can be used to manipulate the objects.

Assignments

An Assignment has the form

destination = source

Destination of an assignment can be a data field or a variable.
Source of an assignment can be:

- A data field Ex.: **:100 = :200**
- A variable Ex.: **:100 = var**
- A string constant Ex.: **:100 = "Text"**
- A parameter Ex.: **:100 = &P2**
- The result of a call to a built-in function

Deletion of a data field will be done by an assignment of an empty string. Ex.: **:100 = ""**

Set (error) message

The script can return a message to the calling program (f.e. the cat client) by the MESSAGE-statement.

MESSAGE [:field-tag] "number" [+ "Text"]...

The optional parameter **:field-tag** attaches the message to a dedicated field.

"number" is the message number which corresponds to the text of the message in the file **message.xxx** (xxx used for language). If the message has variable parts, these parts can be appended by the chararcter **+**.

Ex.:

```
MESSAGE :100 "1120" attaches the message with number 1120 to the field 100.
```

Procedure calls

Recurring statement sequences can be assembled in a subordinate script (procedure). A procedure starts . like the script itself - with

PROC procname

and ends with

END PROC

The procedure is called by

```
DO(procname)
```

The calling statement can deliver up parameters to the procedure. Then the call is done by

```
DO(procname(parm1, parm2, ...))
```

Inside the procedure the parameters are addressed by **&P1**, **&P2**, ...
There is no limit of the amount of parameters.

Control statements

Control statements control the logical flow of statement sequences. Control statements and their statement sequences can be nested unlimited.

IF

The IF-clause controls the execution of statements depending on a condition.

IF condition THEN statements [ELSE statements] END IF

Condition is the result of a comparison of two objects.

Compare operators are **=**, **<**, **<=**, **>**, **>=** and **#** (stands for not equal).

Multiple conditions can be combined with the logical operators **AND** and **OR**.

Is the condition is true, the statements after **THEN** will be performed.

Is the condition is false, the statements after **ELSE** will be performed, if there are any.

All statements until **END IF** (resp. ELSE) will be performed.

Examples:

```
IF :100 = "YES" THEN :200 = "" END IF
```

This statement deletes field 200 if the contents of field 100 is YES.

WHILE

The statements **WHILE** and **LOOP** define loops of statements. The WHILE-clause repeat a statements sequence until the accompanying condition is true.

WHILE condition statement END WHILE

Condition is the result of a comparison of two objects. Compare operators are =, <, <=, >, >= and # (means not equal). More than one condition can be combined with the logical operators AND and OR.

Compare operators are **=**, **<**, **<=**, **>**, **>=** and **#** (stands for not equal).

Multiple conditions can be combined with the logical operators **AND** and **OR**.

As long as the condition is true, the statements until **END WHILE** will be performed.

LOOP The LOOP-clause is like the WHILE-clause, but it tests the condition at the end of the statement sequence. So the statements were executed at least once.

LOOP statements UNTIL condition

CHOOSE

If it is necessary to execute several alternative statements depending on the content of an object, you can use the CHOOSE-clause.

CHOOSE source CASE condition statements [CASE condition statements]... END CHOOSE

source is the object which should be compared. condition is the belonging comparison operation. If it is true the following statements are executed until the next **CASE** or until **END CHOOSE**.

Ex.:

```
CHOOSE :100
CASE = "A"
  (statements, if content of field 100 is A)
CASE = "B"
  (statements, if content of field 100 is B)
CASE = "C"
  (statements, if content of field 100 is C)
END CHOOSE
```

Built-in functions

The built-in functions can be used to modify data fields. The functions can have data fields, constant strings, integers, variables or function calls as parameters.

BEGSTR(s, t)	t is a string: if the string s starts with the string t, the function result is t t is a string set: if the string s starts with one of the strings in t, the function result is that string
CHKFLD(s, x)	checks the string s depending on the string x: if x = "ISBN" Formal ISBN check if x = "ISMN" Formal ISMN check if x = "ISSN" Formal ISSN check if x = "DATE" Formal check of a valid date of form YYYYMMDD If the check is okay, the function results in an empty string, otherwise an error code
CONCAT(s, t)	concatenates strings s and t in the result
DELSTR(s, t)	t is a string: removes string t from string s in the result t is a string set: removes all strings in t from string s in the result
DELSTR(s, t, u)	t and u are strings: removes that string from s which starts with string t and ends with string u and returns the result t and u are string sets: removes those strings from string s which start with one of the strings in t and end with the corresponding string in u and returns the result
ENDSTR(s, t)	if t is a string: results in t, if the string s ends with the string t t is a string set: if the string s ends with one of the strings in t, the result is that string
INITCAP(s)	The result is string s, but the first letter will be uppercase, the others lowercase. (Only letters a - z, A - Z are converted)
INSTR(s, t)	is a string: if the string t is included in the string s, the result is t t is a string set: the result is that string in t which is included in the string s
INSTR(s, t, u)	t and u are strings: the result is that substring of s which starts after the string t and ends before the string u t and u are string sets: the result is that substring of s which starts after one of the strings in t and ends before the corresponding string in u (the strings t and u are not included)
LEFT(s, t)	t is a string: : the result is the part of string s which is left from string t

	t is a string set: the result is the part of string s which is left from one of the strings in t
LOWER(s)	the result is string s in lower case characters
LPAD(s, n, t)	results in string s, filled from left side with the string t up to length n
LTRIM(s)	results in string s without leading spaces
LTRIM(s, t)	t is a string: results in string s without leading strings t
	t is a string set: results in string s without all leading strings in t
NL	results in newline
REPLACE(s, t, u)	t and u are strings: replaces string t in string s with string u in the result
	t and u are string sets: replaces all strings in t in string s with the corresponding string in u
RIGHT(s, t)	t is a string: the result is the part of string s which is right from string t
	t is a string set: the result is the part of string s which is right from one of the strings in t
RPAD(s, n, t)	results in string s, filled from right side with the string t up to length n
RTRIM(s)	results in string s without tailing spaces
RTRIM(s, t)	t is a string: results in string s without tailing strings t
	t is a string set: results in string s without all tailing strings in t
SUBSTR(s, n)	results in that substring of s, starting in column n
SUBSTR(s, n, m)	results in that substring of s, starting in column n and ending in column n + m
UPPER(s)	the result is string s in upper case characters (Only letters a - z, A - Z are converted)
ADD(i, j)	for integer values : returns i + j
SUB(i, j)	for integer values : returns i - j
MUL(i, j)	for integer values : returns i * j
DIV(i, j)	for integer values : returns i / j (if j not equal zero)
DATE(f)	returns the actual date in format f. Following formats are allowed: DD.MM.YYYY DD.MM.YY MM/DD/YYYY MM/DD/YY YYMMDD YYYYMMDD YYDDD YYYYDDD
TIME(f)	returns the actual daytime in format f. Following formats are allowed: HH.MM.SS HH.MM HH:MM:SS

Examples:

```
STRING s = "This is a text"
```

Call	**Result**
BEGSTR(s, "This")	"This"
CHKFLD("1-111-11111-1", "ISBN")	""
CONCAT(s, ", too")	"This is a text, too"
DELSTR(s, "a ")	"This is text"
ENDSTR(s, "text")	"text"
INITCAP("abcdefg")	"Abcdefg"
INSTR(s, "is")	"is"
INSTR(s, "This", "text")	"is a "
LEFT(s, "a")	"This is"
LOWER(s)	"this is a text"
LPAD(s, 25, "xyz")	"xyzxyzxyzxyThis is a text"
LTRIM(" xyz")	"xyz"
REPLACE(s, "a", "not a ")	"This is not a text"

RIGHT(s, "a")	" text"
RPAD(s, 25, "xyz")	"This is a textxyzxyzxyzxy"
RTRIM(s, "xyz ")	"xyz"
SUBSTR(s, 9)	"a text"
SUBSTR(s, 9, 3)	"a t"
UPPER(s)	"THIS IS A TEXT"
DATE("DD.MM.YYYY")	"26.09.2001"
TIME("HH:MM:SS")	"10:39:55"
ADD(5, 2)	7
SUB(5, 2)	3
MULT(5, 2)	10
DIV(5, 2)	2

Comments

Comments are started by //. All characters after // until the end of line are treated as a comment.