Digitals Script Language

Functionality of the toolbars (Window|Create Toolbar) was improved. It is possible to use variables and formulas in the new version of the script language. Conditional expressions and cycles allow you to create the script programs that can execute group processing both at the level of objects and at the level of the opened maps. Now it is also possible to use the functions for calling dialogs to create interactive scripts.

Variables

Variables are the memory cells named by user. They can store values returned by functions and they can also be sent to the functions as parameters. All variables must start with **$** symbol and can consist of Latin characters, digits and underline. Preliminary variable declaration is not required and variable "starts to work" as far as it appears in the text of the script.

Example:

$C=@Map.Count	the number of objects of the active map is assigned to $C variable
@Map.SelectObject $C	the last object of the map is selected

Functions

Functions (commands) allow to execute various operations with the map's objects and to receive the information about their current state. Function always starts with **@** symbol and can have input parameters which follow its name through space. For example, for the function **@Map.SelectObject** the parameter is the number of object you want to select.

The call of the function **@Map.SelectObject 1** will select the first object.

Some functions do not execute any operations, but only return values, for example, **@Map.Count** returns the number of objects of the map, and **@Map.NextSelected** returns the number of the selected object (or zero value if there are no selected objects). For further usage of the value returned by function it should be set to the variable.

Example:

$Name=@Map.ClearFilename	sets the filename (without extension) of the map to $Name variable
@Map.SaveToFile $Name.dxf	saves the map to the file of DXF format

Expressions

Expressions serve to set to the variables the values that are calculated on the basis of numerical constants and values of other variables.

Example:

$A=1	sets value 1 to the variable $A
$B=$A+5	stores in $B variable the sum of $A variable and numerical value 5
$B=$B+1	increase by one the value of $B variable

(As the result of this script, the variable $A will contain value 1, and $B value 7)

Labels

Labels serve to assign the name to any line of the program and allow to make the conditional or unconditional jumps to a certain place of the program by means of the **@Goto** function. All labels must start with **%** character and located on a separate line. They are usually used for cycle organization.
Example:

$Count=MapCount	sets the quantity of the opened maps to the variable $Count
@If $Count=0 @Break	break the script if there are no opened maps
$I=1	sets the initial value 1 to the cycle variable $I
%Start	label – the start of the cycle
$Name=Map[$I].ClearFilename	filename (without ext) for map $I assigned to the variable $Name
@Map [$I].SaveToFile $Name.dxf	saves the map to DXF file format (with the same name)

$I=$I+1	increases the number of the processed map by one
@If $I<=$Count @Goto %Start	return to Start if the map's number is less than or equal to $Count

As the result of this script, all opened in Digitals maps will be saved as the files with DXF extension.

Conditions

The function **@If** is used to check the condition. It can be used in the reduced and the expanded format. The reduced format looks like the following:

@If Expression Operation

Operation can be any function (expression) that will be executed if expression is true.

Example:

@If $C=0 @Break calls @Break (script break) if $C value is zero

The expanded format of the **@If** function allows to use more difficult expressions and to set operation which is executed if the expression is false. The expanded format looks like the following:

@If Expression then Operation1 else Operation2

Example:

@If ($I>0) and ($I<=$Count) then @Goto %Next else @Goto %Start

If **$I** value is greater than zero and less than or equal to **$Count,** passes to a label **%Next** (continue execution), otherwise passes to a label **%Start** (for example gets back to data entry).

Dialogs

Dialogs are the special functions which allow the script to co-operate with the user, requesting him the necessary data that can be set to the variables for further usage. In the current version the following dialogues are realized:

Dialog.Ask Text

Displays a window with an input field and returns the value entered by the user.

Example: **$Tol=@Dialog.Ask Enter tolerance value**

If the user presses button **Cancel** or key **Esc** script execution will be broken and if he presses button **OK (Enter)** the entered value will be set to the variable **$Tol.**

Dialog.Confirm Text

Shows a window with buttons **OK** and **Cancel** to confirm an action.

Example: **$C=@Dialog.Confirm Delete the object?**

If the user presses button **OK** the variable **$C** will return value 1, otherwise it will return value 0.

Dialog. Message Text

Outputs a text box with the message.

Example: **@Dialog.Message Process is completed successfully**

Dialog. Select Title|Value1|Value2 |...

Outputs a window with the choice of the value from the list.

Example: **$A=@Dialog. Select Set the rotation angle 90|180|270**

If the user presses the button **Cancel** or key **Esc** script execution will be broken, otherwise the entered value will be set to the variable **$A**.

Dialog.OpenFile Filter Filename

Opens a dialogue with the choice of the file for opening. Optional parameter Filter sets a mask and optional parameter Filename sets the default filename.

Example:

$Name=Dialog.OpenFile
$Name=Dialog. OpenFile *.in4

$Name=Dialog. OpenFile *.txt C:\Digitals\1.txt

If the user presses the button Cancel or key Esc the function will return null value, otherwise the full name of the selected file will be set to the variable **$Name**. To check if the value is null you can use **@If** function, e.g. **@If $Name= then @Break.**

Dialog. SaveFile Filter Filename

Opens the dialog with a choice of filename to save. Optional parameter Filter sets a mask and optional parameter Filename sets the default filename.

Example:

$Name=Dialog. SaveFile *.dxf
$Name=Dialog. SaveFile *.txt C:\Digitals\1.txt

If the user presses the button Cancel or key Esc the function will return a null value, otherwise the full name of the selected file will be set to the variable **$Name**. To check if the value is null you can use **@If** function, **e.g. @If $Name= then @Break.**

Remarks

The Remark is any line of the script that starts with the **;** symbol (semicolon). Remarks do not participate in the script execution and can be used to insert the explanations or disconnect some lines from the executable file.

Arrays

Arrays are another kind of functions that allow directly address different ordinal (listed) Digitals objects. Access to the concrete array cell is made by its number - the index. The index of an array cell is specified in square brackets. Arrays can be nested, so that the array cell of higher level can contain the arrays of lower level. E.g. every cell of the array **Map** contains the array of map objects, and every object in the same way contains the array of object parameters**.**

In the current version of the program the following arrays are realized:

Map[N]

The array **Map** contains all the maps opened in the program. To get the quantity of maps (windows) opened at present moment use the **MapCount** function (not the **Map.Count** function, which returns the quantity of objects in the active map). While working with one (an active) map its index can be omitted.

E.g., to save an active map use the function **Map.Save** and to save the map opened in the second window use the function **Map[2].Save**.

All the functions that have the prefix **Map** can be also used with the prefix **Map[N],** executing certain actions with any opened map and not only with an active one.

Map[N].Object[N]

The array **Object** contains all the objects of the map. To get the quantity of objects in the map use the function **Map[N].Count.** To find out the quantity of points in the object use the function **Map[N].Object[N].Count.**

E.g. **Map[1].Object[1].Count** will return the quantity of points in the first object of the first opened map, and **Map.Object[2].Count** - the quantity of points in the second object of the active map.

Map[N].Object[N].Point[N]

The array **Point** contains all the points of the map's object. The example, this is the script that saves all the points of the selected object in the text file:

```
$Sel=@Map.NextSelected
@if $Sel<=0 @Break Please, select the object
$C=@Map.Object[$Sel].Count
$I=1
%Start
$P=@Map.Object[$Sel].Point[$I]
@Text.Add $P
$I=$I+1
@If $I<=$C @Goto %Start
@Text.Save C:\test.txt
```

The array **Point** can be used not only to get the point co-ordinates but also to set the co-ordinates to any point.

E.g. **@Map.Object[$Sel].Point[1] $P** will set the coordinates of the variable $P to the first point.

Besides access to all three co-ordinates, it is also possible to get/set values of/to each of the three co-ordinates (X,Y,Z) separately, e.g.:

$X=@Map[1].Object[1].Point[1].x

or

@Map[1].Object[1].Point[1].z 0

Map[N].Object[N].Parameter[N]

The array **Parameter** keeps the values of all the parameters of the map's objects. Unlike the other arrays, parameters can be addressed to not only by their number, but also by the ID and the identifier In4.

Example:

```
$Value=@Map.Object[1].Parameter[0]
$Value=@Map.Object[1].Parameter[ID10000]
$Value=@Map.Object[1].Parameter[CN]
```

Just like the points, the values of the parameters can be both read and set, e.g.

@Map.Object[1].Parameter[ID10000] John Smith or **@Map.Object[1].Parameter[ID30] $NONE**

(Built-in variable $NONE is used to set the empty value)

Text[N]

The **Text** array is intended for work with text lists (text files). Each array cell is the arbitrary set of text lines, which can be added to, changed in or saved at the file and loaded from it. Within the limits of one script it is possible, if necessary, to use up to 63 different text files addressed to as **Text[1], Text[2], Text[63].** If only one text file is used in the script, it can be addressed to as **Text** without the index.

Example:

```
@Text.Load C:\test.txt
$Value=@Map.Object[1].Parameter[3]
Text.Add $Value
@Text.Save C:\test.txt
```

To work with the files the following commands are available:

Text[N].Load Filename – loads the text from the text file

Text[N].Save Filename – writes the text to the text file

Text[N].Add Text string – adds a text string to the end of the file

Text[N].Count – returns the quantity of the lines in the file

Text[N].Clear – clears the file

Text[N].Line[N]

The array **Line** is intended to access every string of the file by its number (starting with the first one). The value of the string can be read (stored to the variable) and written (set the new value from the variable or in the direct way).

Example:

```
@Text.Load C:\test.txt
$L=@Text.Line[1]
$Value=@Map.Object[1].Parameter[ID30]
@Text.Line[1] $L $Value
```

If you want to set the empty value to the string use built-in variable $NONE, e.g **@Text.Line[3] $NONE.**

Other new functions

Break Text

Breaks the script execution and shows a window with the message if optional parameter **Text** is present.

CheckErrors 1/0

Enables/disables built-in error handler. By default the handler is enabled and in case of the error the script breaks with the error message. In the script editing mode the line with the last error will be highlighted in red.

This script will be aborted on second line with error message and Dialog.Message command won't be executed:

```
$I=0
@Map.SelectObject $I
@Dialog.Message Script is finished
```

And in this case the error will be ignored and the script execution will be continued to the end:

```
@CheckErrors 0
$I=0
@Map.SelectObject $I
@Dialog.Message Script is finished
```

In you disable built-in error handler you'd better do all the necessary checks of the variable values, input data etc. The function **@CheckErrors** can be used repeatedly and error handling can be enabled/disabled during the script execution.

Map.ClearFilename

Returns full name of the file without its extension. E.g. for the map stored in the file **C:\Digitals\MyMap.dmf** the function will return **C:\Digitals\MyMap** string.

@Calc function

@Calc calculates result of formulas with mathematical expressions or text string operations. The function has the following syntax **$S=@Calc Expression**. Math and string functions/operators that can be used in the expression are listed bellow:

Variable types

```
  x,y         : numeric - (integer, float)
  a,b         : boolean (1 or 0)
  s,t,v       : string
  d           : DateTimeString  (StampString)
```

Basic operations

```
  numeric:          x + y , x - y , x * y, x / y, x ^ y
  compare:          x > y, x < y, x >= y, x <= y, x = y, x <> y
  ansi compare:     s > t, s < t, s >= t, s <= t, s = t, s <> t
  boolean (1/0):    a AND b,  a OR b,  NOT(a)
  set variable :    x:=formula (or value) ;
  destroy variable: FreeVar(s);    // s=variable name
  logical:          ExistVar(s)  // s=variable name
  formula separation with semicolon :  formula1 ; formula2
```

Type conversion

```
  boolean (1/0):   Logic(x)
  numeric:         Numeric(s)
  string:          String(x)
  char:            Char(x)
  integer:         Ascii(s)

  all types:       Eval(f)  // where f is formula in [...]
  string :         NumBase(x,base) // base from <2..16>
  integer:         BaseNum(s,base) // base from <2..16>
```

Math operations

```
numeric (integer): x Div y,  x Mod y
```

Math functions

```
   Abs(x), Frac(x), Trunc(x), Heaviside(x) or H(x), Sign(x),
   Sqrt(x), Ln(x), Exp(x),
   Cos(x), CTg(x), Ch(x), CTh(x), Sin(x),  Sh(x), Tg(x), Th(x),
   ArcSin(x), ArcCos(x), ArcTg(x), ArcCtg(x),
   MaxVal(x [,y, ...]),  MinVal(x [,y, ...]),
   SumVal(x [,y,...]),   AvgVal(x [,y, ...])
```

String operations

```
   s || t ,
   s Like t,      // (%,_)
   s Wildcard t   // (*,?)
```

String functions

```
  integer: Length(s), Pos(t,s)
  string:  Trim(s), TrimLeft(s), TrimRight(s), Upper(s), Lower(s),
           Copy(s,x,[y]), CopyTo(s,x,[y]), Delete(s,x,[y]),
           Insert(s,t,x);
           Replace(s,t,v,[1/0=ReplaceAll,[1/0=IgnoreCase]] );
           IFF(a,s,t);    //IF a>=1 then Result:=s else Result:=t
```

Date & Time functions

```
  integer: Year(s), Month(s), Day(s), WeekDay(s),
           Hour(s), Minute(s), Sec(s)
  numeric: StrToStamp(d)  LastDay(x) // last day in Month (28-31)
  string:  StampToStr(x), StampToDateStr(x), StampToTimeStr(x)
```