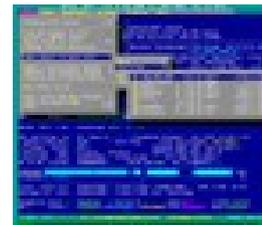


# *TxWin native scripting*

Jan van Wijk

TxWin native scripting language  
as available in DFSee 9.03 and up,  
and TxWin starting from version 2

**FSYS** - *software*



**TxWin**

# *Presentation contents*

- DFSee scripting history, design goals and alternatives
- High level layout of TxScript programs
- Script parameters and variables
- Expressions, available operators
- Built-in functions
- Control structures, program flow
- Expression substitution in commands
- Example scripts fragments, from DFSee usage

# *Who am I ?*

## Jan van Wijk

- Software Engineer, C, Rexx, Assembly, PHP
- Founded FSYS Software in 2001, developing and supporting DFSee from version 4 to 14.x
- First OS/2 experience in 1987, developing parts of OS/2 1.0 EE (Query Manager, later DB2)
- Used to be a systems-integration architect at a large bank, 500 servers and 7500 workstations
- Developing embedded software for machine control and appliances from 2008 onwards

Home page: <https://www.dfsee.com/>

# *Dfsee scripting history*

- Over time, to automate repeating and more complex tasks, several scripting methods have been (and still are!) used with DFSee:
  - BAT/CMD/SHELL scripts, calling DFSee
  - Rexx subcmd environment for the OS/2 version
  - Native scripting, being a simple list of DFSee commands, executed sequentially, with simple error handling and parameter substitution

# *TxScript design goals*

- Backwards compatible with existing .DFS scripts as far as possible, allowing re-use
- Direct access to much DFSee internal info, including disk sectors from a script
- Powerful expressions, variables and functions  
Can be used from and in the (DFSee) command-line too
- Conditional and looping control to allow more intelligent and powerful scripts

Note: For 'DFSee' you can read any hosting program that uses the TxScript engine from the TxLib library

# *Do we need another language ?*

- Trying to avoid re-inventing yet another wheel, some alternatives have been considered:
  - Rexx, as used in OS/2 version already
  - Python, clean OO type language
  - Perl, very powerful, hackers heaven :-)
  - PHP, Ruby etc as used in WEB environments
- All had problems with integration in the hosting program (DFSee), availability on all required platforms, or added complexity for install etc.
- Developing a new language is fun, so YES :-)

# *High level layout of TxScript*

- LINE-oriented, but ignores whitespace usage within and between lines. Each line is either:
  - A comment line (ignored mostly :-)
  - An interpreter 'pragma' altering its behaviour
  - Program flow statements like IF or WHILE
  - An assignment to one or more script variables
  - A command to be passed to the host (DFSee) to be executed, including substitution of expressions

# *Example for script layout*

```
;script example
```

```
;;defaultparam 1 5
```

```
IF $1 < $_parts
```

```
    Say $1 is OK!
```

```
ENDIF
```

- A comment line
- A pragma
- Control statement with an expression
- A command to be executed by DFSee
- End of the Control statement

# *Script parameters and variables*

- Parameters to the script are positional, and named \$1 through \$9, \$0 is the scriptname
- Variables follow the 'Perl' syntax where possible, with a subset of the functionality
  - \$variable a scalar variable
  - \$array[index] scalar taken from an array
  - %array whole array
  - \$hash{key} scalar taken from a hash
  - #hash whole hash variable

# *System variables*

- Variablenames starting with '\$\_' are system variables (DFSee) and are read-only
  - They come as scalar and scalar-from-array variants
- Some examples (there are dozens :-)
  - `$_parts` total number of partitions, 1..n
  - `$_disk` current opened disk number
  - `$_this` sector number for current sector
  - `$_d_size[X]` size in sectors for disk nr X
  - `$_p_fsform[Y]` FS-format for partition nr Y
  - `$_b_sector[Z]` Contents of sector nr Z, in a (512 byte) binary string

# *Expression and variable values*

- Variable and expression values are either:
  - A string of arbitrary length, may contain any character value from 0..255, allowing binary data manipulation
  - A 64-bit signed integer value, allowing huge numbers while maintaining the exact integer value
- Expression operators and built-in functions automatically convert between these
  - Other types like floating-point may be added later

# *Expressions, operators, functions*

- Expression syntax and semantics are pretty close to those defined in 'Perl' and 'C' but are not exactly identical
- Operators work on 1, 2 or 3 operands:
  - Unary, like + - ! NOT 1 operand
  - Binary, like + \* < = 2 operands
  - Ternary, (cond) ? exp1 : exp2 3 operands
- Textual operators like 'AND' must be uppercase!
- Functions take zero or more arguments and return a value (in an expression)

# Operator precedence, high to low

\$name[]++ --

Atom, Term

- + ! ~

\* / %

+ -

X

.

<< >>

== != < > <= >=

=== !==

EQ NE LT GT LE GE

- Variable, indexed, auto increment/decrement
- String, number, function nested-expr or ternary
- Unary operators
- Binary multiply/division
- Binary plus/minus
- String replication
- String concatenation
- Numeric bit-shift
- Numeric compare
- Same value AND type
- String compare

# Operator precedence, part 2

&

^

|

- Bitwise AND
- Bitwise XOR
- Bitwise OR

&&

||

- Logical AND (C-style)
- Logical OR (C-style)

=

,

- Assignment
- Comma, multi-expression

NOT

AND

OR

- Logical NOT (Perl style)
- Logical AND (Perl style)
- Logical OR (Perl style)

# *Built-in functions, A-F*

abs  
b32  
b2asc  
b2int  
chr  
canceled  
confirmed  
defined  
drivefs  
drivelabel  
drives  
drivespace  
exists  
filext  
fnbase

- Absolute value, numeric
- Clip to 32-bit unsigned
- Binary string to ASCII
- Binary string to reversed int
- ASCII value for number
- Test for canceled last operation
- Confirmation Yes/No/Cancel
- Is variable defined
- FS-name for drive letter
- Label string for drive letter
- All drive letters in string
- Freespace in KiB for drive
- File exists
- Set default file extension
- Extract filename without ext

# *Built-in functions, G-M*

fnfile  
fnpath  
getcwd  
h2asc  
h2int  
i2dec  
i2hex  
index  
lc  
left  
length  
mkdir  
max  
min  
message

- Extract filename without path
- Extract path only, no filename
- Get current working directory
- Get string from hex-ascii str
- Get integer from hex-ascii str
- Convert int to decimal str
- Convert int to hexadecimal str
- Find substring in string
- Return lowercased string
- Left adjust string, pad/clip
- Get length of string
- Create full directory path
- Ret maximum of values
- Ret minimum of values
- Message popup, until [OK]

# Built-in functions, O-Z

ord  
prompt  
replace  
sec2gib  
sec2kib  
sec2mib  
reverse  
right  
rindex  
strip  
  
substr  
uc  
undef

- Numeric value 1<sup>st</sup> char in str
- Popup question, return string
- Replace characters in string
- Get GiB value for #sectors
- Get KiB value for #sectors
- Get MiB value for #sectors
- Reverse characters in string
- Right adjust string pad/clip
- Reverse find substring in str
- Strip leading/trailing chars from a string (default spaces)
- Extract substring from string
- Return uppercased string
- Undefine (free) a variable releasing any used storage

# Control structures, branching

IF (condition)

statement-list

- Like the Perl IF, not using a {} block but an ENDIF keyword

ELSEIF (condition)

statement-list

- () parenthesis on conditions optional

ELSE

statement-list

- Any number of the ELSEIF clause

ENDIF

- ELIF, ELSIF and ELSEIF accepted

# Control structures, looping

WHILE (condition)  
statement-list  
ENDWHILE label

FOR init;condition;iterator  
statement-list  
ENDFOR label

DO label  
Statement-list  
UNTIL (condition)

- 'C' like, explicit END replaces any {} block
- () parenthesis on conditions optional
- 'break' exits the loop, can take a 'label' too
- 'continue' skips code upto the loop iterator
- Labels are optional

# Control structures, more looping

LOOP

Statement-list

ENDLOOP

- Endless loop, no condition at all

LOOP

EXIT label WHEN (cond1)

Statement-list

IF (condition2)

Statement-list

break label

ENDIF

Statement-list

EXIT label WHEN (cond3)

ENDLOOP label

- LOOP with one or more exit conditions at arbitrary positions
- Mainly useful when using the LABELS in nested loops :)

# Command expression substitution

- Transparent, replacing expressions by the result of the expression, when starting with a variable:
  - `$_this + 100`
  - `Wipe z $start $_d_cylsize * 25`
  - `Say You have $_parts partitions on $_disks disks`
- Explicit, enclose in **double** curly brackets if NOT starting with a variable, or any conflicting syntax:
  - `Restore {{$imgfile}} -P:$partition ; -P conflicting`
  - `Say we are in: {{getcwd()}} ; not a variable`

# *Miscellaneous comments*

- Keywords are case-insensitive (IF, WHILE)
- Parenthesis on conditions are optional
- Conditions must be on a single line, or use explicit line continuation
- Lines are 'continued' using '\' as last char allowing long expressions to be spread over more than one physical line

# *Miscellaneous comments*

- Script syntax is checked BEFORE running any statement, except expressions to be substituted in commands (to be refined :-)
- Single '\$' characters in commands will be left 'as-is' so can be used freely, but when directly followed by any alphabetic a-z/A-Z it could be mistaken for a variable and you need to escape that by doubling the '\$' character as: '\$\$'
- There may be application level mechanisms too, that allow switching variable substitution on/off. Would result in better readable commands ...

# *Considered improvements*

- User defined functions or subroutines
- More/better array and hash variable handling and manipulation (perl like)
- Floating point variables
- Basic file-I/O, read/write text and binary

# *Example code fragments - 1*

Set default parameters, in named variables

```
;;defaultparam 1 0 ;disk to work on, 0 = auto
;;defaultparam 2 '$0' ;default image name
;;defaultparam 3 2 ;minimum number of disks
;;defaultparam 4 99 ;maximum number of disks
;
log $0 ;same as scriptname
$stick = $1
$image = $2
$dmin = $3
$dmax = $4
$stickmsg = "bootable multi-ISO, (USB) disk"
```

# *Example code fragments - 2*

Check DFSee version and number of disks

```
if $_version >= 1000
  if ($_disks >= $dmin) && ($_disks <= $dmax)
    ; ... do the real work ...
  else
    confirm Need $dmin to $dmax disks, got: $_disks
  endif
else
  confirm Script needs DFSee 10.x (this is $_version)
endif
```

# *Example code fragments - 3*

Get size + number smallest accessible disk  
(taken from the DFSUSB32.DFS script)

```
$size = 999999999
for $disk = 1; $disk <= $_disks; $disk++
    if $_d_size[ $disk] < $size
        if $_d_access[ $disk]
            $stick = $disk
            $size = $_d_size[ $stick]
        endif
    endif
endfor
```

# Example code fragments - 4

;Create a FAT32 partition on a memory stick

```
cr -d:$stick pri fat32 -M -o -L:"-v:Sdata -p:Stick2 -l:*"  
if $_rc == 0  
  'format' -f:32 -v:DfStickdata  
  if $_rc == 0  
    lvm -n:DFSeeUSBStickBIG -d:$stick  
    $exitmsg = FAT32 created and formatted."  
  else  
    $exitmsg = Create FAT32 partition failed!"  
  endif  
endif  
part -d:$stick
```

# *Example – recovery script core*

```
confirm -y Recreate $parts partitions on disk $work
if $_rc == 0
    $done = 0
    while (1) ;single pass, allow break from section
        ; ... multiple recovery sections here (see next slide)
        break
    endwhile
    part -d -n
    if $done == $parts
        confirm $done partitions done~~Press a key to exit
    endif
else
    confirm Recovery canceled by user
endif
```

# *Example – recovery script section*

;add one section for every partition, with specific message

```
cr pri bmgr 1 -a:0,c -F -l-
```

```
if $_rc == 0
```

```
  $done++
```

```
else
```

```
  confirm Create partition $done +1 failed $abortmsg
```

```
  break
```

```
endif
```

```
cr log hpfs 2000 -at:6001,c -L:"-v:eCS -p:Boot -l:C -menu"
```

```
if $_rc == 0
```

```
  $done++
```

```
else
```

```
  confirm Create partition $done +1 failed $abortmsg
```

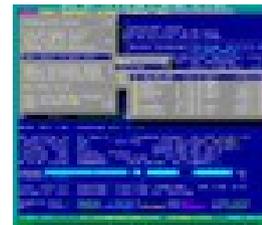
```
  break
```

```
endif
```

# *TxWin native scripting*

## Questions ?

**FSYS** - *software*



**TxWin**