What's "Dana Script" ?

"Dana Script" is a programmable macro language for the text editor "Dana". It is very similar to BASIC language in syntax. On the other hand, it has some specifications like C language.
Most of control structures, numerous functions and properties to control Dana, plus calling DLL functions capability, can help you to write flexible and powerful macro scripts for Dana.

Language specification

Symbol description

<arg>	parameter
<var>	variables
<statement>	statement
<declare>	declare statement
<name>	symbol name
<expr>	expression
<const>	constant value
<const string>	constant string
<type>	type declaration
[...]	described between [] can be omitted.

Data Type

The data types that Dana Script supports are Integer(4 byte binary), String and their arrays.
It does not support the data types such as 'Single', 'Double', 'Variant', and 'Type'.
'Long' is not implemented because it means the same as Integer.
Integer ranges from -2,147,483,648 to 2,147,483,647.
String type has no limitation in size, but constant string in source code is limited to 256 bytes.

Operators

Arithmetic operator
- + Plus or concatinate strings.
- - Minus
- / Division integer
- * Multiply integer
- Mod Remainder after Division.

Logical operator
- | Logical OR
- & Logical AND
- ^ Logical XOR

Relational operator
- Or
- And
- Xor

Comparing operator
- > Left is greater than right.
- >= Left is greater than or equal to right.
- = Left is equal to right.
- <= Left is less than or equal to right.
- < Left is less than right.
- <> Left is not right.

Single operator
- + Positive.
- - Negative.
- Not Non zero to zero and zero to non zero.

Bit operator
- << Shift to Left
- >> Shift to Right

Declaration

Declaration of variables and procedures are necessary in Dana Script.

Proc
Const
Dim, Static
Declare
Option

Proc

```
Proc <name> (<arg>, <arg>, ...)
    <declare>
    <statement>
    [Return <expr>]
End Proc
```

Proc statement declares an internal procedure. Prototype like C language is not required. Declaration of an internal procedure is also an entry point of it. And one 'Proc Main()' is necessary for a module.
(You can omit 'Proc' statement for 'Main' procedure)
A script always begins with the Main procedure regardless of actual location in source code.

e.g.

```
Proc foo(arg1, arg2)
    Return (arg1 + arg2)
End Proc
```

Const

Const <name> = <const>
Const <name> = <const string>

Const statement declares a constant value. You can declare both Integer type and String type in the same way.
Especially when the same constant string appears many times in your source code, Const declaration saves a memory consumption.

```
Const A = 1
Const B$ = "ABC"
        -> Actually, you don't have to add $ after symbol.
```

Dim, Static

```
Dim <name> [As <type>]
Static <name> [As <type>]
```

Declaration of variables is absolutely necessary in Dana Script.
When you declare variables using "Dim" inside of a procedure, they are defined as "Local" variables (usable only inside of a procedure).
When you declare them outside of a procedure, they are defined as "Global" variables. (It is usable in everywhere after the declaration in a source code.)
Variables are filled by zero initially.

When a type-description is omitted, it is assumed as the Integer type.

e.g.

```
Dim A
Dim B$
Dim C(10)
Dim D$(20,20)
```

When you want to declare array of variables, specify the size of array in ().
You can declare multi-dimensional array stating the size of array delimitering by ','.
Index of array begins by 1 as default.

Static statement is pretty much the same as Dim if it is used outside of the procedure.
It is different when used inside of the procedure however. Static variables keep their values even after exitting the procedure.

e.g.

```
'Each time "foo" is called, 'S' increments by 1.
'"A" is initialized to zero each time at the entry of this procedure ,
'so the result of "A" should be always 1.
Proc foo()
    Static S
    Dim A
    S = S + 1
    A = A + 1
    MsgBox("S is " + Str$(S))
    MsgBox("A is " + Str$(A))
End Proc
```

Declare

Declare Proc <name> Lib "lib" [Alias "alias"] (<arg>,<arg>,[..]) [As <type>]

"Declare" statement is used for declaring a procedure in DLL. Dana Script supports only 32 bit DLL including, of course, Win32 API.

"lib" is the file name of DLL. ".DLL" extention can be omitted.
"alias" is the actual procedure name in DLL. If <name> is the same name as actual DLL exported name and it doesn't have any problem(for example exported name conflicts Dana Script's reserved word), it can be omitted.
When you describe ".." at the end of parameter list, it allows "variable" parameters. This enables you to call C type declared function that has variable parameters such as "wsprintf".

e.g.

```
Declare Proc wsprintf Lib "User32" Alias "wsprintfA" (szArea$, szFmt$, ..)
```

In Dana Script, not like ordinary BASIC, a parameter is passed "By Value". If you want to pass an address of a variable, see Tips.

e.g.

```
' Windows API call sample
' Menu prompts you to select whether "Iconic" or "Maximize".
' If you select "Iconic" then Window will be minimized,
' Otherwise Window will be maximized.
Const SW_SHOWMINIMIZED = 2
Const SW_SHOWMAXIMIZED = 3

Declare Proc ShowWindow Lib "User32" (hWnd%, nShowCmd%)

Main ()
    Dim hMenu
    hMenu = NewMenu()
    AddMenuItem(hMenu, "&Iconic",    SW_SHOWMINIMIZED)
    AddMenuItem(hMenu, "&Maximize", SW_SHOWMAXIMIZED)
    ' hMainWnd property is Dana's main window handle.
    ShowWindow(hMainWnd, DoMenu())
    DiscardMenu(hMenu)
End
```

Option

"Option" statement is to control Dana Script compiler. Following two options are supported.

Option Base <const>
Set array index base. it normally should be 0 or 1.
Default value is 1.
Option Stack <const>
Set stack size. Usually you don't have to specify this value.
Default value is 8192 bytes(2048 level).

Control structure

Dana Script supports most of control structures those are supported by some other structured programming language.(C, PASCAL etc.)

If
Do .. Loop
Select Case
While .. Wend
For .. Next
Return
Exit

If

If Statement

```
If <expr1> Then
    <statement1>
[Else If <expr2> Then
    <statement2>]
[Else
    <statement3>]
End If
```

When expr1 is True then execute statement1.
If expr1 is not True and expr2 is True then execute statement2.
Otherwise execute statement3.

Sometimes a expression is too long and you might want to devide it by entering return like this....

```
If ..
Then
    ..
Else
If ..
```

But be patient. "If .. Then" or "Else If .. Then" must be written in one line.

```
If <expr> Then <statement>
```

In this case, you can't use "Else" or "End If" statement.

Do .. Loop

```
Do While|Until <expr>
    <statement>
Loop
```

This statement repeats executing <statement> while <expr> is True(Do While) or False(Do Until).

```
Do
    <statement>
Loop While|Until <expr>
```

This statement repeats executing <statement> while <expr> is True(Loop While) or False(Loop Until).
<statement> is executed at least one time.

Select Case

```
Select Case <expr0>
Case <expr1>
     <statement1>
[Case <expr2>
     <statement2>]
[Case Else
     <statement3>]
End Select
```

When <expr1> is matched to <expr0> do <statement1>, and when <expr2> is matched to <expr0> do <statement2> otherwise do <statement3>.

While .. Wend

```
While <expr>
      <statement>
Wend
```

It is pretty much the same as "Do While .. Loop".

For .. Next

```
For <var> = <expr1> To <expr2> [Step <const>]
    <statement>
Next [<var>]
```

"For" statement repeats <statement> incrementing <var> by 1(or specified value by "Step") while <var> is growing from <expr1> to <expr2>.
Be sure that it not to be an endless loop.

Return

"Return" statement is to be used when you want to exit the procedure.
If you want to return a value, use Return statement with an expression.

e.g.

```
   Return True                    'True is as the return value.
```

Exit

"Exit" statement is used when you want to exit a current loop block or a procedure.
You can write it like "Exit Do", "Exit For", but it just make the readability of your source code better. No actual influence.

e.g.

```
  For I = 1 To 10
      J = I
      Do While True
          J = J - 1
          If J < 0 Then Exit       ' Absolutely exit to 1>.
      Loop                           ' Even if you write it "Exit For",
1>                                   ' never exit to 2>.
  Next I
2>
```

Calling procedures

There are 3 types of procedure in Dana Script:
User defined procedure, built in procedure and DLL procedure.

Whatever the procedure type is, you can call it in following 3 ways.

1) rc = Foo(Arg)
2) Call Foo(Arg)
3) Foo(Arg)

1 is the function calling. You can get a return value by calling function in this way. And you can use the return value as an argument of the other function like this.

e.g.

```
If Baa(Foo(Arg)) = True Then ...
```

2 and 3 are the procedure calling. There is no actual difference between them. Use whichever you prefer. If procedure has a return value, it is just ignored.

BASIC-LIKE built-in statement is not supported in Dana Script. Some alternative functions are available instead.

e.g.

```
Ordinary BASIC statement:      Open "file" For Input As #1
Dana Script:                   fp = FOpen("file", "r")
```

*Additional notes about Built-In functions

In particular cases, you can omit certain parameters. See the explanation of each functions to check how it works when you omit the parameters.

*Additional notes about DLL calling.

For general information, see the explanation of Declare statement.
Followings are some additional information.

1) If you want to pass an address of the return area, you can pass String variable.
You must have allocated enough space for this string before the call.

e.g.

```
Dim str1$
str1$ = Space(22)
DLLProc(str1$, 20)      'DLLProc returns string to str1$
```

2) "" doesn't mean NULL. If you want to pass NULL, pass zero.

Command Script and Resident Script

Dana Script supports two forms of executable.
One is the Command Script. It works like parmanent editting command of Dana.
The other is the Resident Script. It stays on memory and responds to certain events posted by Dana.
Actually, there is no essential difference. Only the way of implementation is different.

Command Script
Resident Script

Command Script

The implementation of Command Script is very simple.
"Main" procedure entry is the start of the script, and the corresponding "End" is the end of the script. There is no other rule.

Resident Script

The entry point of Resident Script is the "Main" procedure like Command Script. But Resident Script should always check ".DanaState" property and dispatch to a procedure suitable for the current event.
If you are familiar with the Windows SDK programming, it may be quite easy to understand. Usually you can use TsrTmpl.DAS as a template.

As default, Dana.DAS is loaded as a Resident Script when you start Dana. Adding code on it, you can customize the default actions of Dana.

".DanaState" can be one of following values.
Practically, you should declare in your source code as follows.

```
Const STATE_INIT         = 0      'Normally executed
Const STATE_BEFORE_OPEN  = 1      'Before open file.
Const STATE_AFTER_OPEN   = 2      'After open file.
Const STATE_AFTER_NEW    = 3      'After new file.
Const STATE_BEFORE_SAVE  = 4      'Before save current file.
Const STATE_AFTER_SAVE   = 5      'After save current file.
Const STATE_BEFORE_CLOSE= 6      'Before close current window.
Const STATE_BEFORE_EXIT  = 7      'Before exitting editor.
Const STATE_KEY_PRESS    = 8      'Certain key was pressed except
character key.
Const STATE_CHAR         = 9      'Certain character is inputted.
Const STATE_ENTER        = 10     'Enter key is pressed.
Const STATE_TIMER        = 11     'Per second.
Const STATE_ACTIVATE     = 12     'Current window is activated.
```

When you receive STATE_INIT, you should call "StayResident" procedure to stay on the memory. And at a certain timing you like, you can call "Terminate" procedure to release this.

Parameters are set in ".ParmA", ".ParmB", ".ParmStrA", "ParmStrB" properties.
Its meaning depends on the value of ".DanaState".

STATE_INIT

STATE_BEFORE_OPEN
    .ParmStrA = File name to open.
    Return:
        .ParmA <- 0 I don't wanna open this file.

STATE_AFTER_OPEN

STATE_AFTER_NEW

STATE_BEFORE_SAVE
.ParmStrA = File name to save.
Return:
.ParmA <- 0 I don't wanna save this file.

STATE_AFTER_SAVE

STATE_BEFORE_CLOSE

STATE_BEFORE_EXIT

STATE_KEY_PRESS
.ParmA = Virtual key code.
.ParmB = Shift status.
Virtual key code is compatible with Windows' virtual key code.
For more information, see appendix Virtual key codes
Shift status represents a combination of Ctrl, Shift, and Alt keys.
Just as follows.

Ctrl	&H20
Shift	&H40
Shift+Ctrl	&H60
Alt	&HFE
Shift+Alt	&HFD

Return:
.ParmA <- 0 I don't want this key affects Dana.

STATE_CHAR
.ParmA = Ascii character code
Return:
.ParmA <- 0 I don't want to input this character.

STATE_ENTER
Return:
.ParmA <- 0 I don't want to input return.

STATE_TIMER

STATE_ACTIVATE

Additional information of language specification.

* Error trapping is not supported.
  You should check return values if the procedure returns error information.
* Delimiter of multi statement is ";"(semicolon).
* If parameter is Integer, value is passed. If String, pointer is passed.
* Internal string specification is ASCIIZ (terminated by zero).
* GoTo statement is not supported. <g>

Built-In functions

Conversion
Abs
Asc
Chr
Hex
Str
Val
RGB

Abs
Abs(nVal%) As Integer

Parameters:
    nVal%   Value to be converted to absolute value.
Return:
    Converted value
Description:
Converts a specified value to an absolute value.

Asc
Asc(sStr$) As Integer

Parameters:
    sStr$ String.
Return:
    Character code for the first character in a string.

Description:
Returns a character code (Integer) for the first character in passed string.

Chr
Chr(nVal%) As String

Parameters:
    nVal%   Character code
Return:
    One character string whose ASCII code is the argument.

Description:
Converts a character code to a character string.
It is useful to represent non-printable characters or some special characters like "(double quote).

Example:
    Chr(&H22) + "Test" + Chr(&H22) + Chr(&H0A)   --> "Test"<LF>

Hex
Hex(nVal%) As String

Parameters:
    nVal%   Integer value
Return:
    Hex string

Description:
Converts a Integer value to a string which represents hexadecimal.

Str
Str(nVal%) As String

Parameters:
    nVal%   Integer value
Return:
    Decimal string.

Description:
Converts a Integer to a string which represents decimal value.

Val

Val(sStr$) As Integer

Parameters:

sStr$ A string represents decimal value.

Return:

Decimal value which is represented by the string parameter.
If conversion is failed, it returns zero.

Description:

Converts a string, which represents decimal value, to numeric decimal. If you want to pass hexadecimal string, put "&H" prefix at the top of a string.

JisToSjis
JisToSjis(nJis%) As Integer

Parameters:
Return:
Description:

SjisToJis
SjisToJis(nSjis%) As Integer

Parameters:
Return:
Description:

RGB
RGB(nRval%, nGval%, nBval%) As Integer

Parameters:
nRval% Red value(0 - 255)
nGval% Green value(0 - 255)
nBval% Blue value(0 - 255)
Return:
Long value which represents color.

Description:
Returns color expression made from R,G,B information.

String manipulation

InStr
Len
Left
Right
Mid
LTrim
RTrim
Trim
Space
LCase
UCase

InStr
InStr(sTarg$, sFind$) As Integer

Parameters:
- sTarg$ String expression to be searched.
- sFind$ String expression to be sought.

Return:
- The position at which sFind$ is found within sTarg$.
- If not found, returns zero.

Description:
Returns the position one string is found within another string.

Len
Len(sStr$) As Integer

Parameters:
    sStr$ String.
Return:
    Length of sStr$

Description:
Returns the length of a string.

Left
Left(sStr$, nLen%) As String

Parameters:
sStr$ String.
nLen% Length of return characters.
Return:
Leftmost string.
Description:
Returns the leftmost nLen% characters of a string argument.

Right
Right(sStr$, nLen%) As String

Parameters:
sStr$ String.
nLen% Length of return characters.
Return:
Rightmost string.

Description:
Returns the rightmost nLen% characters of a string argument.

Mid
Mid(sStr$, nBeg%, nLen%) As String

Parameters:
    sStr$ String.
    nBeg%   Beginning position.
    nLen%   Extracting length.
Return:
    Extracted string.

Description:
Returns a string extracted from a string parameter. The extracting position is indicated by nBeg% and the length is indicated by nLen%.

LTrim
LTrim(sStr$) As String

Parameters:
    sStr$ String
Return:
    Trimmed string

Description:
Eliminates tabs or whitespaces from left side of a string.

RTrim
RTrim(sStr$) As String

Parameters:
    sStr$ String.
Return:
    Trimmed string.

Description:
Eliminates tabs or whitespaces from right side of a string.

Trim
Trim(sStr$) As String

Parameters:
    sStr$ String.
Return:
    Trimmed string.

Description:
Eliminates tabs or whitespaces from both sides of a string.

Space
Space(nLen%) As String

Parameters:
    nLen%   Numbers of spaces you want.
Return:
    A string of whitespaces.

Description:
Returns a string which contains whitespaces with the specified length.
This function is also useful to allocate dummy area for string variable.

LCase
LCase(sStr$) As String

Parameters:
    sStr$ String to convert.
Return:
    Converted string.

Description:
Returns a string whose characters are converted to lower case.

UCase
UCase(sStr$) As String

Parameters:
    sStr$ String to convert.
Return:
    Converted string.

Description:
Returns a string whose characters are converted to upper case.

Input and Output

Eof
FOpen
FGetInt
FGets
FPutInt
FPuts
FSeek
FClose
FCopy
FKill
Dir

Eof
Eof(nFp%) As Integer

Parameters:
    nFp%   File pointer.
Return:
    True   Already reached to EOF
    False   Not reached to EOF yet.

Description:
Checks if the file pointer has already reached to the end of the file.

FOpen
FOpen(sFile$, sFlg$) As Integer

Parameters:
sFile$ File name to open.
sFlg$ File open mode. it can be "r"(Read), "w"(Write) or "rw"(Read Write (as default)).
Return:
Non 0 File pointer.
0 Failed to open.

Description:
Opens a specified file with a specified opening mode.
File pointer which is returned can be used for calling another file handling function. When you don't want to use the file pointer any more, you have to close it by calling FClose function.

FGetInt
FGetInt(nFp%, nBytes%) As Integer

Parameters:
nFp% File pointer.
nBytes% Bytes to read.(1 to 4)
Return:
Numeric value read from file.

Description:
Returns a numeric value read from a file by specified bytes. Number of bytes which you can specify should be up to 4(bytes of integer).
This function is useful for binary file handling.

FGets
FGets(nFp%) As String

Parameters:
    nFp%   File pointer.
Return:
    A string read.
    "" means EOF

Description:
Returns a string read from a file. End of line is converted to LF even if it is CRLF. If you don't need LF code, truncate it using Left() function.

Example:
```
    str1$ = FGets(nFp)
    MsgBox(Left(str1$, Len(str1$)-1))
```

FPutInt
FPutInt(nFp%, nBytes%, nVal%) As Integer

Parameters:
nFp% File pointer.
nBytes% Bytes to write.(1 to 4)
nVal% Value to write.
Return:
True Succeeded to write.
False Failed to write.

Description:
Writes numeric value by specified bytes. If the value isn't representable in specified bytes, over bytes is truncated.
This function is useful for binary file handling.

FPuts
FPuts(nFp%, sStr$) As Integer

Parameters:
    nFp%   File pointer.
    sStr$ String.
Return:
    True   Succeeded to write.
    False   Failed to write.

Description:
Writes string to file. If you want a carriage return code, add LF code at the end of the string. Do not add CR code.

Example:
```
    FPuts(nFp%, "Test" + Chr(10))               'LF(Chr(10)) can be converted
                                                'CRLF code finely.
    FPuts(nFp%, "Test" + Chr(13) + Chr(10)) 'Another CR code(Chr(13))
                                                'is added. Not good.
```

FSeek
FSeek(nFp%, nLen%, nMode%) As Integer

Parameters:
nFp% File pointer
nLen% Number of bytes to move
nMode% Seek mode
0 From top of file.
1 From bottom of file.
2 From current position of file pointer.
Return:
Absolute position of file pointer after moved.

Description:
Moves position of the file pointer to specified position in the file.

FClose
FClose(nFp%)

Parameters:
nFp% File pointer
Return:
None

Description:
Closes the file. You have to close a file opened by FOpen().

FCopy
FCopy(sSrc$, sTarg$) As Integer

Parameters:
    sSrc$ File name to copy.
    sTarg$ File name to be copied.
Return:
    True   Copied successfully.
    False   Failed to copy.

Description:
Copies specified file to another.

FKill
FKill(sFile$) As Integer

Parameters:
sFile$ File name to delete
Return:
True Deleted successfully.
False Couldn't be deleted.

Description:
Deletes specified file.

Dir
Dir(sMask$) As String

Parameters:
    sMask$ File name or file pattern.
Return:
    File name that was found.
    "" means specified file was not found.

Description:
Returns the file name that matches to the specified file pattern. It never returns directory name.

Dana Command - General

Command
KeyToCmd

Command
Command(sCmd$)

Parameters:
sCmd$ Command name of Dana.(Case sensitive)
Return:
None

Description:
Calls permanent command of Dana. For information about the command names, see the appendix "Commands of Dana

KeyToCmd
KeyToCmd(nKey%, nSft%) As String

Parameters:
nKey% Virtual key code.
nSft% Shift status.
Return:
Command name

Description:
Retrieves command name from a particular key bind(virtual key code and shift status).
Virtual key code is compatible with Windows' virtual key code.
For more information, see appendix Virtual key codes
Shift status represents a combination of Ctrl, Shift, and Alt keys, as shown below.

Ctrl	&H20
Shift	&H40
Shift+Ctrl	&H60
Alt	&HFE
Shift+Alt	&HFD

Dana Command - Edit

AddString
Convert
Sort
CopyToFile
PasteFromFile
AppendFile

AddString
AddString(sAdd$, nPos%)

Parameters:
sAdd$ String to add.
nPos% Position at which you are adding.
0 Top of string.
1 End of string.
Return:
None

Description:
Adds specified string at the specified position of each selected string.

Convert
Convert(nCase%, nTab%, nDBCS%)

Parameters:
nCase% Case conversion mode.
0 No conversion.
1 ALL TO UPPER.
2 all to lower.
3 Top of line to upper.
4 Top Of Word To Upper.
nTab% Tab conversion mode.
0 No conversion.
1 Tab to Space.
2 Space to Tab.
Return:
None

Description:
Converts selected string with specified conversion mode.

Sort
Sort(nOrder%, nBlkKey%)

Parameters:
nOrder% Sort order
0 Ascendantly(as default).
1 Descendantly.
nBlkKey% Box sort mode
0 Sort only inside of the box(as default).
1 Sort entire lines treating the box selection as sort keys.
Return:
None

Description:
Sorts selected strings.

CopyToFile
CopyToFile(sFile$) As Integer

Parameters:
    sFile$ File name to save.
Return:
    Always True (in this version)

Description:
Writes selected strings to a specified file.

PasteFromFile
PasteFromFile(sFile$) As Integer

Parameters:
    sFile$ File name to read
Return:
    Always True (in this version)

Description:
Reads a specified file at the caret position.

AppendFile
AppendFile(sFile$) As Integer

Parameters:
    sFile$ File name
Return:
    Always True (in this version)

Description:
Appends selected strings at the end of a specified file.

Dana Command - File

FileOpen

SaveAs

FileOpen
FileOpen(sFile$, nROnly%) As Integer

Parameters:
sFile$ File name to open.
nROnly% Read only flag.
0 Not read only (as default)
1 Open as read only file.
Return:
Always True (in this version)

Description:
Opens a specified file to edit.

SaveAs
SaveAs(sFile$) As Integer

Parameters:
    sFile$ File name to save
Return:
    Always True (in this version)

Description:
Saves the current work file as a specified file name.

Dana Command - Find

FindFor
FindBack
Replace
Grep

FindFor
FindFor(sFind$, sOpt$) As Integer

Parameters:
sFind$ String to find.
sOpt$ Option string (contains following characters.)
G Search from the top of text.
A Search in all windows currently opened.
M Mark found line.
I Ignore case
W Match whole word.
T Match top of string only.
E Match end of string only.
R Use regular expression.
If not specified anything, default find option is used.
Return:
True Found.
False Not found.

Description:
Searches specified string forward.

FindBack
FindBack(sFind$, sOpt$) As Integer

Parameters:
Return:
see FindFor

Description:
Searches specified string backward.

Replace
Replace(sFind$, sRepl$, sOpt$) As Integer

Parameters:
sFind$ String to find.
sRepl$ String to replace.
sOpt$ Option string (contains following characters.)
G Search from top of text.
A Search in all windows currently opened.
M Mark replaced line.
I Ignore case.
W Match whole word.
T Match top of string only.
E Match end of string only.
R Use regular expression for finding.
N Replace all without confirming.
If not specified, default replace option is used.
Return:
True Found string to replace at least one.
False Not found any string to replace.

Description:
Replaces specified string with another one. This function returns True as long as one string has been found to replace, even if you canceled replacing.

Grep
Grep(sFind$, sDir$, sMask$, sOpt$) As Integer

Parameters:
sFind$ String to Find
sDir$ Directory to search.
sMask$ Target file pattern.
sOpt$ Option string (contains following characters.)
I Ignore case.
W Match whole word.
T Match top of string only.
E Match end of string only.
R Use regular expression for finding.
U Search sub directories recursively.
P Output file name as full path.
If not specified, default Grep option is used.
Return:
Always returns True.

Description:
Searches a specified string in files on your disk, and when found, outputs the result strings formatted with tag style to the Output Screen of Dana.
Tag style is like as follows.

FILENAME.TXT(1):
FILENAME.TXT(5):
FILENAME.TXT(20):
FILENAME.TXT(23):

Dana Command - Jump
JumpLine

JumpLine
JumpLine(nLineNo%, nMode%)

Parameters:
nLineNo% Line number.
nMode% Line number count mode.
0 As logical(count of return code)(as default)
1 As shown (including folded line without return code)
Return:
None

Description:
Goes to a certain line whose line number is specified.

Dana Command - Exec

ShellCmd

ShellMenu

ShellCmd
ShellCmd(sCmd$, sOpt$) As Integer

Parameters:
sCmd$ Command line.
It can include following macro symbols.
%F File name of current file.
%N File name without extention.
%P Full path name of the current file.
%D Directory name of the current file.
%E{Env} String indicated by environment string in {}
%T Name of a temporary file to which selected string is saved.
%I Prompt user to input a string here.
%% '%' itself.
sOpt$ Option string(contains following characters)
I Run minimized.
O Get console output to the Output Screen.
E Get console output to the caret position.
C Run via command shell.
Return:
Always True (in this version)

Description:
Runs the other application.

ShellMenu
ShellMenu(nCmd%) As Integer

Parameters:
    nCmd%   Program number in the Louncher(0 to 25)
Return:
    Always True (in this version)

Description:
Runs the other application registered in the Launcher of Dana.

Edit
GetCurrentLine
GetNext
GetNextMark
GetPrev
GetPrevMark
GetThisLine
GetTopLine
LoadThisLine
SaveThisLine
GetCount
GetToThisCount
GetCurrentChar
GetCursorWord
GetSelected
DelSelect
SelectCancel
IsMarked
SetMark
IsModified
SetModified
InputChar
InsertString
SetCursorPos
GotoThere

GetCurrentLine
GetCurrentLine() As Integer

Parameters:
None
Return:
Line Handle.

Description:
Retrieves a line handle of the current line. Lline handle is a unique value that represents a certain line, and is used for the other line-handling functions.

GetNext
GetNext(hLine%) As Integer

Parameters:
    hLine% Line handle.
Return:
    Next line handle.

Description:
Returns the line handle of the line next to the specified line.

GetNextMark
GetNextMark(hLine%) As Integer

Parameters:
    hLine% Line handle.
Return:
    Next marked line handle.

Description:
Returns the line handle of the next marked line to the specified line.

GetPrev
GetPrev(hLine%) As Integer

Parameters:
    hLine% Line handle.
Return:
    Previous line handle.

Description:
Returns the line handle of the line previous to the specified line.

GetPrevMark
GetPrevMark(hLine%) As Integer

Parameters:
    hLine% Line handle.
Return:
    Previous marked line handle.

Description:
Returns the line handle of the marked line previous to the specified line.

GetThisLine
GetThisLine(nLineNo%) As Integer

Parameters:
nLineNo Line number.
Return:
Line handle.

Description:
Retrieves the line handle of the line specified by the line number.

GetTopLine
GetTopLine() As Integer

Parameters:
None
Return:
Line handle.

Description:
Retrieves the line handle of the top line of the text.

LoadThisLine
LoadThisLine(hLine%) As String

Parameters:
    hLine% Line handle
Return:
    String.

Description:
Retrieves the actual string represented by the line handle.

SaveThisLine
SaveThisLine(hLine%, sStr$) As Integer

Parameters:
    hLine% Line handle.
    sStr$ String to restore.
Return:
    Updated line handle.

Description:
Replaces the string represented by the line handle with specified string.
The passed line handle is not proper after call this function, so use returned new line handle as line handle for that line.
Note that this function clears undo/redo buffers.

GetCount
GetCount(hLine%) As Integer

Parameters:
hLine% Line handle
0 means current line(as default)
Return:
Count of turned line.

Description:
Returns the count of turned (continued from the right side of the window to the left side without carriage return.) lines in the current logical line.
For instance, if not turned, it returns 1. If turned once, it returns 2.

GetToThisCount
GetToThisCount() As Integer

Parameters:
None
Return:
Position from logical top of line.

Description:
Returns the count from the logical top of the current line to the caret position.
For instance, if the logical top of the line and the caret position is matched, it returns 0.

GetCurrentChar
GetCurrentChar() As Integer

Parameters:
    None
Return:
    Character code.

Description:
Returns an ASCII character code on the caret position.

GetCursorWord
GetcaretWord() As String

Parameters:
    None
Return:
    A word on caret.

Description:
Returns a delimitered word on the caret position.

GetSelected
GetSelected() As String

Parameters:
None
Return:
A string copied selected region.

Description:
Returns the selection as a string. Return code is converted to CRLF.
If the selection mode is box, each end of line is converted to CR and end of block is CRLF.

DelSelect
DelSelect()

Parameters:
None
Return:
None

Description:
Deletes the text which is currently selected .

SelectCancel
SelectCancel()

Parameters:
None
Return:
None
Remarks:
Cancels the selection of the current window.

IsMarked
IsMarked(hLine%) As Integer

Parameters:
hLine% Line handle
0 means current line (as default).
Return:
True Marked.
False Not marked.

Description:
Checks if a specified line is marked or not.

SetMark
SetMark(bMark%, hLine%)

Parameters:
bMark% True Mark.
False Don't mark.
hLine% Line handle
0 means current line(as default).
Return:
None

Description:
Marks a specified line.

IsModified
IsModified() As Integer

Parameters:
None
Return:
True Already edited.
False It is not edited yet.

Description:
Checks if the current file is modified or not.

SetModified
SetModified(bModify%)

Parameters:
bModify% True Set editted flag.
False Clear editted flag.
Return:
None

Description:
Sets/Resets the modified flag of the current file.

InputChar
InputChar(nChar%)

Parameters:
    nChar% ASCII character code.
Return:
    None

Description:
Inputs a character represented by an ASCII character code to the caret position.

InsertString
InsertString(sStr$)

Parameters:
    sStr$ String to insert.
Return:
    None

Description:
Inputs a string to the caret position.

SetCursorPos
SetcaretPos(nCsrX%, nCsrY%)

Parameters:
    nCsrX   X position.
    nCsrY   Y position.
Return:
    None

Description:
Moves caret to the specified XY position.

GotoThere
GotoThere(nLineNo%, nColm%)

Parameters:
nLineNo% Line number as shown (0 means don't move line).
nColm% Column position (0 means don't move column).
Return:
Nothing.

Description:
Moves caret to the specified position in the current text.

System
NewMenu
AddMenuItem
DoMenu
GetMenuItem
DiscardMenu
MsgBox
InputBox
GetOpenFile
Beep
DoEvents
Environ
AppActivate
SendKeys
ShowCursor
Time
Date
Wait

NewMenu
NewMenu() As Integer

Parameters:
None
Return:
Menu handle.

Description:
Creates an empty menu. You must call DiscardMenu to release the handle after used.

AddMenuItem
AddMenuItem(hMenu%, sItem$, nID%) As Integer

Parameters:
hMenu% Menu handle
sItem$ Menu item string.
nID% Menu ID (as you like).
But following two is reserved.
0 Cascaded Menu.
65535 Separater.
Return:
True Successfully added.
(or sub menu handle, when you specified nID% as 0)
False Failed to add.

Description:
Adds a menu item to the menu.
You can create cascaded menu specifying nID% as 0. In this case, the returned value is a handle of a new menu which you can use to add a sub menu item.
nID% must be a unique value in one menu including sub menu, except 0 and 65535.

DoMenu
DoMenu(hMenu%) As Integer

Parameters:
    hMenu% Menu handle
Return:
    Menu item ID that was selected.
    65535 means cancelled.

Description:
Executes the menu and returns the item ID that identifies each menu item.

GetMenuItem
GetMenuItem(hMenu%, nID%) As String

Parameters:
    hMenu% Menu handle
    nID%   Item ID.
Return:
    Menu item string.

Description:
Returns the menu item string that is identified by the item ID.

DiscardMenu
DiscardMenu(hMenu%)

Parameters:
    hMenu% Menu handle.
Return:
    None

Description:
You have to call this function to release the menu handle that will not be used any more.

MsgBox
MsgBox(sMsg$, sTitle$, nStyle%) As Integer

Parameters:
sMsg$ Message string.
sTitle$ Title string of message box (omittable)
nStyle% Style of message box(omittable)
(Compatible with MessageBox;Windows API function.)

Return:
(Compatible with MessageBox;Windows API function)

Description:
Opens a message box dialog. Some constant values (like "MB_YESNO","IDOK" and so on) are not defined in Dana Script permanently, so you should define these constants using "Const" statement (or use constant numeric as a parameter directly).
For more information, see appendix Other Constants

InputBox
InputBox(sMsg$, sTitle$, sDefault$) As String

Parameters:
sMsg$ Message string.
sTitle$ Title string (omittable).
sDefault$ Default string(omittable).
Return:
Inputted string.

Description:
Opens a dialog box for inputting a string and returns the string which is input by the user.

GetOpenFile
GetOpenFile(Filt$, bPath%, bSave%) As String

Parameters:
    Filt$ A filter string which specify the file type.
    bPath% If True, returns full path name.
    bSave$ If True, open "Save As" dialog instead of "Open" dialog.
Return:
    File name
Remarks:
Opens the common dialog box of file open and returns a specified file name.
Returning a file name is only the purpose of this function (Do nothing with it).

Beep
Beep()

Parameters:
    None
Return:
    None

Description:
Beep for attention.

DoEvents
DoEvents()

Parameters:
None
Return:
None

Description:
Flushes all the messages that are queued in the Windows' message queue.

Environ
Environ(sEnv$) As String

Parameters:
    sEnv$ Environment string.
Return:
    A string indicated by the environment string.

Description:
Returns a string that is indicated by the environment string.

AppActivate
AppActivate(sTitle$, sClass$) As Integer

Parameters:
    sTitle$ Window title. "" Matches all windows.
    sClass$ Window class name. "" Matches all windows.(omittable)
Return:
    Non 0   Succeeded to activate.

Description:
Activates the application whose window matches the specified window title and the window class.
"Window title" parameter is not necessarily matched to a whole title string. For example, if sTitle$ is "Dana", it matches the following window title.

Dana - C:\FILE.TXT

IMECtrl
IMECtrl(nMode%)

Parameters:
Return:
Description:

SendKeys
SendKeys(sKeys$, bWait%)

Parameters:
sKeys$ Key string
% ALT key
^ Ctrl key
+ Shift key
special keys like "Up", "Down" should be inside {}
For more information about special keys,
see appendix "Special keys.
bWait% If True, wait all key strokes has been processed.
Return:
None

Description:
Sends key strokes to the active application window.
This function enables you to control the other application using with AppActivate function.

Example:
```
SendKeys("%FO")       'Alt+F("File"menu)->"Open"
SendKeys("^Tab")   'Ctrl+Tab
```

ShowCursor
Showcaret(bShow%)

Parameters:
bShow True Enables to show a caret.
False Disables to show a caret.
Return:
None

Description:
Enables/Disables to show a caret.

Time
Time(sFormat$) As String

Parameters:
sFormat$ Format string (omittable)
%H hour(00 - 23)
%I hour(01 - 12)
%M minute(00 - 59)
%p AM/PM
%S second(00 - 59)
%% % itself

Return:
A string represents current time.

Description:
Returns a time string. You can use a format string to get time information in any form as you like.
For instance, if you describe like Val(Time("%H")), you can get the hour information as numeric decimal.

Date
Date(sFormat$) As String

Parameters:
sFormat$ Format string (omittable).
%y year(00 - 99)
%Y year(Long)
%m month(01 - 12)
%b short month name
%B long month name
%d date(01 - 31)
%a short week name
%A long week name
%% % itself

Return:
A string represents current date.

Description:
Returns a date string.

Wait
Wait(nTime%)

Parameters:
    nTime% Time for wait.(ms)
Return:
    None

Description:
Halts for a specified period of time.

Execute
Run
Shell

Run
Run(sScript$)

Parameters:
    sScript$ Script name. ".DAS" extention is omittable.
Return:
    None

Description:
Runs another script.

Shell
Shell(sCmdLine$) As Integer

Parameters:
    sCmdLine$ Command line.
Return:
    True   Succeeded to execute.
    False   Failed to execute.

Description:
Executes the other program. You can also use "ShellCmd" function as well, but in this function, you can specify a document name whose extention is related to a certain application.

Example:
```
  Shell("Test.XLS")
```

Special
LodB
StoB
Alloc
ThrowAway

LodB
LodB(sStr$, nIdx%) As Integer

Parameters:
    sStr$ String.
    nIdx%   Index.
Return:
    ASCII character code.

Description:
Returns a character at the specified index position of the string.
Be careful to use this function because it does not check the validation of the address pointed by the index.
The first character of the string is pointed by index 0.

StoB
StoB(sStr$, nIdx%, nChar%)

Parameters:
    sStr$ String.
    nIdx%   Index.
    nChar%   ASCII character code.
Return:
    None

Description:
Restores a character at the specified index position of the string.
Be careful to use this function because it does not check the validation of the address pointed by index.
The first character of the string is pointed by index 0.

IsKanji
IsKanji(nChar%) As Integer

Parameters:
Return:
Description:

Alloc
Alloc(nSize%) As String

Parameters:
    nSize% Size to allocate.
Return:
    String type variable that is allocated by specified size.

Description:
Allocates space for a string variable. You don't have to use this function usually.

ThrowAway
ThrowAway(sMem$)

Parameters:
    sMem$ String variable
Return:
    None

Description:
Releases a buffer which is allocated for a string variable. You don't have to use this function usually.

Others
GetMarkFile
GotoNext
Silent
NoSilent
Refresh

GetMarkFile
GetMarkFile() As String

Parameters:
None
Return:
Mark file name.

Description:
Returns the mark file name for the current work file.

GotoNext
GotoNext(nMode%) As Integer

Parameters:
nMode% Moving mode
0 Go to the next document. (as default)
1 Go to the next window that has same document.
Return:
True Moved successfully.
False Failed to move.

Description:
De-activate the current window and activate another window.

Silent
Silent()

Parameters:
None
Return:
None

Description:
Stops drawing window.

NoSilent
NoSilent()

Parameters:
None
Return:
None

Description:
Restarts drawing window. You have to call Refresh() function to redraw all of the windows that belong to Dana.

Refresh
Refresh()

Parameters:
None
Return:
None

Description:
Refreshes all the windows that belong to Dana. You have to call this function after you change some property related to the appearance of Dana. Otherwise Dana may not display itself properly.

Properties

Property is the system variable of Dana which can be accessed in Dana Script.
The name of property always begins with "."(period).
If you see "W" after explanation in the following descriptions, it means the property is changeable. And "R" means it is better to call Refresh() function after you change the property. "S" means it is a String type property. "L" means it is effective for the current edit window only.

If you change "R" marked property and don't call Refresh() function, it will be called automatically when you are exitting the current procedure. If you think this timing is OK, you don't have to call Refresh() function each time.

Mode switch

.AddEOF	Add EOF code at the end of a file on saving.	W
.AfterCsr	Permit the cursor to locate beyond the end of a line.	W
.AutoInd	Auto indent mode works.	W
.AutoSplit	Word wrap mode works.	W
.BoxCsr	Display a DOS like box type caret.	W
.CallFiler	Bring up "File Explorer" as "Open" command.	W
.CMode	C language mode works.	W
.DelSpace	Delete whitespace after EOL on saving,	W
.FillTab	In free caret mode, pad tabs from EOL to the caret.	W
.FindSel	Select found string.	W
.FreeCsr	Free caret mode	W
.Insert	Insert mode	W
.JumpMid	Fix caret at the middle of the screen after jumping.	W
.Mirror	Mirroring save is effective.	W
.PhysLine	Display of line numbers is as shown.(not logical)	W
.SaveMark	Save mark information.	W
.ShiftSel	Shift key and direction key is used to select strings.	W
.SoftTab	Input whitespaces instead of a tab code.	WL
.UndoReset	Clear current undo buffers after the file is saved.	W

Appearance

.CrDisp	Show return codes
.DispNum	WR Show line numbers.
.EofDisp	WR Show the end of a file.
.HScroll	WR Show a horizontal scroll bar.
.VScroll	WR Show a vertical scroll bar.
.KeyWords	WR Show keywords in specified colors.
.KeyBold	WR Show keywords in bold font.
.Ruler	WR Show ruler
.StatBar	WR Show the status bar
.TabDisp	WR Show tab
.ToolBar1	WR Show the toolbar 1(File)
.ToolBar2	WR Show the toolbar 2(Edit)
.ToolBar3	WR Show the toolbar 3(Find)
.ToolBar4	WR Show the toolbar 4(Tool)
.ToolBar5	WR Show the toolbar 5(Window/Help)

		WR
.UnderLine	Show an underline on the caret position.	
		WR
.LineSpace	Number of dots between each line.	
		WR
.BigButton	Show buttons of toolbars bigger.	
		WR
.FkeyDisp	Show function keys	
		WR
.LinePtr	Show the line pointer	
		WR

Global control

.AutoSaveS	Seconds to wait for auto backup.	W
.DanaState	Current event of Dana.(Resident Script□j	
.DocNum	Number of files currently opened.	
.hMainWnd	Main window handle of Dana.	
.ParmA	Parameter A.(For Resident Script)	W
.ParmB	Parameter B.	W
.ParmStrA	String parameter A.	W
.ParmStrB	String parameter B.	W
.SaveMode	Save mode.(0:Normal, 1:Tab->Space, 2:Space->Tab)	W
.SprPoint	Word wrap point.	W
.TextMode	Text type for save(0:DOS type,1:UNIX type)	W
.TileMode	Dual tiling mode(0:Maximized,1:Horizontal,2:Vertical)	W
.TrashBin	Number of backup generations	W

Colors

.BakColor	Background color.	W
.CrColor	Return code.	W
.CtlColor	Control code.	W
.EofColor	EOF mark.	W
.LinColor	Line number.	W
.MrkColor	Marked line.	W
.RulColor	Ruler.	W
.RulBkColor	Backgroud of Ruler.	W
.TabColor	Tab code.	W
.TxtColor	Text.	W
.UlnColor	Underscore.	W
.ChgColor	Changing mark.	W

File and Directory

.BackPath	Backup directory.	S
.HomePath	Home directory of Dana.	S
.MirrPath	Mirroring directory.	S
.ModPath	Full path name of Dana.EXE	S
.FileName	File name of the current work file.	LS
.FileTitle	Window title of the current work window.	LS
.FileType	Extention of the current work file.	LS
.PathName	Full path name of the current work file.	LS

Editting

.BlkBeg	Line number of beginning of the selection.	L
.BlkBegC	Column of beginning of the selection.	L
.BlkBegL	Logical line number of beginning of the selection.	L
.BlkEnd	Line number of end of the selection.	L
.BlkEndC	Column of beginning of the selection	L
.BlkEndL	Logical line number of end of the selection.	L
.BlkDisp	Selected string is present.	L
.BlkSel	Now Selecting.	L
.Cols	Column folding position.	
		WL
.Column	Current column.	L
.CsrX	Current caret X position.	L
.CsrY	Current caret Y position.	L
.hWnd	Window handle of current work window.	L
.LineNo	Current line number as shown.	L
.LineNoL	Current logical line number.	L
.ReadOnly	Read only flag.	
		WL
.Shift	Horizontally shifted columns	L
.Tabs	Tab columns	
		WL
.TotLine	Total lines of the current work file.	L

Others

.FindOpt	Find option string. WS
.ReplOpt	Replace option string. WS
.GrepOpt	Grep option string. WS

Appendix

Tips

1)

Dana Script doesn't support "call by reference" and "Type" structure.
But, if you refer the array variable without index, you can get an address of that.
It is only useful as a parameter for certain DLL functions. Because Dana Script has no method to refer a certain address.
For example, it passes the address of the structure consquently.

```
  Declare Proc GetWindowRect Lib "User32" (hWnd%, lpRect%)

  Main()
      Dim Rect(4)
      Dim rct$

      GetWindowRect(hMainWnd, Rect)
      rct$ = "Left,Top of window is " + Str$(Rect(1)) + "," + Str$(Rect(2))
      rct$ = rct$ + Chr$(13) + Chr$(10)
      rct$ = rct$ + "Right,Bottom of window is " + Str$(Rect(3)) + "," + Str$
(Rect(4))
      MsgBox(rct$, "", 0)
  End Proc
```

2)

Dana Script quite looks an BASIC interpleter, but actually it compiles source codes into binary codes for the virtual stack machine of Dana. Therefore every local variables and parameters are on the stack. It means you can apply C like algorithm, like Recursive call, on it.(Maybe... I've never tried that.<g>)

Dana Commands
The following command names can be used as a parameter of Command() function.

Cursor movement
Line Edit
Edit
Scroll
File
Macro
Jump
Find
Tool
Window
Others

Cursor movement
CsrUp
CsrLeft
CsrRight
CsrDown
WordLeft
WordRight
BegLine
EndLine
LeftSide
RightSide

Line Edit
BackSpace
DeleteChar
InputCtrl
InsertTab
ChangeCase
ChgCaseOne
DeleteAfter
DeleteTop
WordBS
WordDel
RepeatOne
WordPaste
TimeStamp
SplitLine
DeleteLine
InsertAft
InsertBef
DupLine
MarkLine
PopLine

Edit
SelectBegin
BoxBegin
BoxPaste
EditPaste
EditCopy
EditCut
Undo
Redo
Repeat
StepDo
EditConvert
EditCenter
EditLeft
EditRight
EditSort
Indent
BackInd
AddString
SelectAll
CopyToFile
PasteFromFile
AppendFile
FlushBuf
SelectClipBd

Scroll
RollUp
RollDown
NormalRlUp
NormalRlDn
SRollUp
SRollDown
AnothRollUp
AnothRollDown
BothRollUp
BothRollDown
PageUp
PageDown
HalfPageUp
HalfPageDown
RollLeft
RollRight
SlowRlup
SlowRldn

File
NewFile
OpenFile
SaveFile
SaveAs
CloseFile
SaveClose
AllSave
AllSaveQuit
ReOpen
Print
PrintPreview
BreakEdit
LockEdit
FileSelect
FileExplorer
AppExit
AllUpdate
AutoSave

Macro
RecordKey
PlayKey
JukeBox

Jump
MakeTop
MakeBot
MakeMid
TextTop
TextBot
JumpLine
PrevPos
NextMark
PrevMark
NextChanged
PrevChanged
MarkList
MarkOff
ScreenTop
ScreenBot
TagJump
BackToTag
OtherParen

Find
SearchFwd
SearchBwd
SearchNext
SearchPrev
Replace
ReplAgain
TitleMark
GetWord
Grep

Tool
Compare
Shell
ShellMenu
LoadWorkspace
SaveWorkspace

Window
Split
TileTwo
TileHorz
TileVert
Cascade
NewWindow
NextPane
AnothWin

Others
ModeInsert
ModeAutoSplit
Menu1
Menu2
Menu3
Menu4
Menu5
AppHelp
Help1
Help2
Help3
Help4
Help5
Addin0
Addin1
Addin2
Addin3
Addin4
Addin5
Addin6
Addin7
Addin8
Addin9
AddinA
AddinB
AddinC
AddinD
AddinE
AddinF
SetProperty

Special keys

RBtn	Left button of mouse.
BS	
Tab	
Enter	
ESC	
Space	
PgUp	
PgDn	
End	
Home	
Left	
Up	
Right	
Down	
Ins	
Del	
F1 - F24	

Virtual Key Codes

VK_RBUTTON	&H02
VK_BACK	&H08
VK_TAB	&H09
VK_RETURN	&H0D
VK_ESCAPE	&H1B
VK_NEXT	&H22
VK_PRIOR	&H21
VK_END	&H23
VK_HOME	&H24
VK_LEFT	&H25
VK_UP	&H26
VK_RIGHT	&H27
VK_DOWN	&H28
VK_INSERT	&H2D
VK_DELETE	&H2E
VK_F1	&H70
VK_F2	&H71
VK_F3	&H72
VK_F4	&H73
VK_F5	&H74
VK_F6	&H75
VK_F7	&H76
VK_F8	&H77
VK_F9	&H78
VK_F10	&H79
VK_F11	&H7A
VK_F12	&H7B
VK_F13	&H7C
VK_F14	&H7D
VK_F15	&H7E
VK_F16	&H7F
VK_F17	&H80
VK_F18	&H81
VK_F19	&H82
VK_F20	&H83
VK_F21	&H84
VK_F22	&H85
VK_F23	&H86
VK_F24	&H87

Other Constants
Message box style (Sometimes sum of these following values)

MB_OK	&H00000000	OK button only
MB_OKCANCEL	&H00000001	OK, Cancel
MB_ABORTRETRYIGNORE	&H00000002	Abort, Retry, Cancel
MB_YESNOCANCEL	&H00000003	Yes, No Cancel
MB_YESNO	&H00000004	Yes, No
MB_RETRYCANCEL	&H00000005	Retry, Cancel
MB_ICONHAND	&H00000010	Hand formed icon.
MB_ICONQUESTION	&H00000020	Question mark.
MB_ICONEXCLAMATION	&H00000030	Exclamation marks.
MB_ICONASTERISK	&H00000040	Asterisks.
MB_DEFBUTTON1	&H00000000	First button is default.
MB_DEFBUTTON2	&H00000100	Second button is default.
MB_DEFBUTTON3	&H00000200	Third button is default.

ƒ□ƒbƒZ□[ƒWƒ{ƒbƒNƒX•Ô‚è’l

IDOK	1	OK selected.
IDCANCEL	2	Cancel selected.
IDABORT	3	Abort selected.
IDRETRY	4	Retry selected.
IDYES	6	Yes selected.
IDNO	7	No selected.