Purpose and signatures of COM server methods.

Wait.mds library documentation, v2.00.

The library is dedicated to the memory of
Nikolai and Raisa Kopiev.

Introduction.

The wait.mds library, to which this description belongs, is a library of automation support based on MS-DOS scripts. The description given here is devoted to the COM server, which is used by the library to perform actions necessary for its functionality. The set of COM server methods, their functionality and parameterization are determined by their use in this library. The server source code is written in VB.NET, therefore the description of its method signatures is given in this language.

Depending on the Windows OS version, the .NET framework versions on each computer differ, and it is also possible for other framework versions to be present on the computer at the user's discretion. For this reason, the compiled COM server file is not included in the library and is supplied as compressed and folded source codes for the purpose of their deployment, compilation and registration on the user's computer using the framework that is installed locally. The COM server is installed by running the installer, which is included in the library source file "wait.mds.bat" and is called from the command line as follows:

call wait.mds.bat /sub:install /install /vb

or by performing complete installation of the library:

call wait.mds.bat /sub:install /install /all

For the installer to work correctly, it is recommended to run the installation under a user with administrator rights on the local computer. If the OS security system is not configured to run console applications, the installer will need to reboot the computer for the settings changes to take effect, after which the installation will need to be run again. Depending on the OS bit depth, the installer automatically installs the 32-bit or 64-bit version of the library. After installation, a subdirectory "wait.mds" will be created in the %ProgramFiles% folder, which will contain the compiled and registered COM server library, as well as its source code and an auxiliary file in VBScript. The contents of the latter file are also included in the COM server source code in VB.NET along with the corresponding methods. For convenience, it is recommended to use the full installation of the library, since in this case the %wait.mds% environment variable is registered for running the main library from any folder, and a web file of its help with a shortcut on the desktop is created. The web help file may be useful when working with the COM server.

The source code of the COM server is functional and can be used on Windows OS from version XP to version 11. Windows XP does not have the .NET framework pre-installed, so Microsoft .NET Framework 3.5 SP1 must be pre-installed on it.

After compilation and registration, the COM server has a public name "WaitMdsApiWrapper". Its use in the source code is no different from other libraries of this type. An example of creating an object and using one of the methods in VBScript:

```
Dim wmo
Set wmo = CreateObject("WaitMdsApiWrapper")
wmo.ScreenShot(0, 0, "C:\snapshot.jpg", True)
```

The above script takes a screenshot and saves it to the file "snapshot.jpg" on the system disk.

List of methods and their description.

Method ActiveWindow()

Returns the numeric value of the current active window handle as a string.

Syntax:

```
ActiveWindow() As String
```

Result:

The handle of the currently active window as a string.

Parameters:

absent

Method ForegroundWindow()

Returns the numeric value of the foreground window handle as a string.

Syntax:

```
ForegroundWindow() As String
```

Result:

The foreground window handle as a string.

Parameters:

absent

Method WindowOfPoint()

Returns the identifier value of the window that contains the specified coordinates.

Syntax:

```
WindowOfPoint(ByVal X As Long, ByVal Y As Long) As String
```

Result:

Window handle as a string.

Parameters:

X — absolute X-coordinate of a point;

Y — absolute Y-coordinate of a point.

Method SetForeground()

Makes the window with the specified handle the foreground window.

Syntax:

```
SetForeground(ByVal hWnd As Long) As Boolean
```

Result:

True if successful.

Parameters:

hWnd — window handle.

Method IsWindow()

Checks whether a numeric value is the identifier (handle) of an existing window.

Syntax:

```
IsWindow(ByVal hWnd As Long) As Boolean
```

Result:

True if successful.

Parameters:

hWnd — window identifier (handle).

Method WindowIsEnabled()

Checks that the window is not locked.

Syntax:

```
WindowIsEnabled(ByVal hWnd As Long) As Boolean
```

Result:

True if the window is not locked.

Parameters:

hWnd — window handle.

Method WindowIsVisible()

Checks whether the window is in the visible state.

Syntax:

```
WindowIsVisible(ByVal hWnd As Long) As Boolean
```

Result:

True if the window is visible.

Parameters:

hWnd — window handle.

Method WindowIsIconic()

Checks whether the window is in a minimized state.

Syntax:

```
WindowIsIconic(ByVal hWnd As Long) As Boolean
```

Result:

True if the window is minimized.

Parameters:

hWnd — window handle.

Method WindowIsZoomed()

Checks whether the window is in a full-screen state.

Syntax:

```
WindowIsZoomed(ByVal hWnd As Long) As Boolean
```

Result:

True if the window is maximized.

Parameters:

hWnd — window handle.

Method WindowIsChild()

Checks whether a window is a child window of another window.

Syntax:

```
WindowIsChild(ByVal hWndParent As Long, ByVal hWnd As Long) As Boolean
```

Result:

True if successful.

Parameters:

hWndParent — parent window handle;

hWnd — child window handle.

Method WindowIsMenu()

Checks if a window is a menu item.

Syntax:

```
WindowIsMenu(ByVal hWnd As Long) As Boolean
```

Result:

True if successful.

Parameters:

hWnd — window handle.

Method WindowIsHung()

Checks whether the window process has become inoperative.

Syntax:

```
WindowIsHung(ByVal hWnd As Long) As Boolean
```

Result:

True if the window does not respond to test messages within 5 seconds, i.e. if the window is in an inoperative state.

Parameters:

hWnd — window handle.

Note:

Does not apply to console windows because they do not have message queue handling.

Method ClientRect()

Gets the absolute coordinates of the window's client area.

Syntax:

```
ClientRect(ByVal hWnd As Long, ByRef ALeft As Long, ByRef ATop As Long, ByRef
ARight As Long, ByRef ABottom As Long) As Boolean
```

Result:

True if successful.

Parameters:

hWnd — window handle;
ALeft — left border of the client area;
ATop — top border of the client area;
ARight — right border of the client area;
ABottom — bottom border of the client area.

Method WindowRect()

Gets the window borders.

Syntax:

```
WindowRect(ByVal hWnd As Long, ByRef ALeft As Long, ByRef ATop As Long,
ByRef ARight As Long, ByRef ABottom As Long) As Boolean
```

Result:

True if successful.

Parameters:

hWnd — window handle;
ALeft — left border;
ATop — top border;
ARight — right border;
ABottom — bottom border.

Method WindowClass()

Returns the window class.

Syntax:

```
WindowClass(ByVal hWnd As Long) As String
```

Result:

String name of the class.

Parameters:

hWnd — window handle.

Method WindowCaption()

Returns the window title.

Syntax:

```
WindowCaption(ByVal hWnd As Long) As String
```

Result:

Window Title Bar.

Parameters:

hWnd — window handle.

Method WindowText()

Returns the window text.

Syntax:

```
WindowText(ByVal hWnd As Long) As String
```

Result:

Window text.

Parameters:

hWnd — window handle.

Method WindowInfo()

Returns a string with window parameters in JSON format.

Syntax:

```
WindowInfo(ByVal hWnd As Long) As String
```

Result:

Window parameters in JSON format.

Parameters:

hWnd — window handle.

Method ShowWindow()

Changes the window visibility settings according to the given command.

Syntax:

```
ShowWindow(ByVal hWnd As Long, ByVal nCmdShow As Long) As Boolean
```

Result:

True if successful.

Parameters:

hWnd — window handle.

nCmdShow — command identifier for changing the visibility of a window. The numerical values of the command correspond to the command values of the Windows API function ShowWindow, they are listed in MSDN Microsoft.

Method MinimizeWindow()

Minimizes the specified window.

Syntax:

```
MinimizeWindow(ByVal hWnd As Long) As Boolean
```

Result:

True if successful.

Parameters:

hWnd — window handle.

Method CloseWindow()

Closes the specified window.

Syntax:

```
CloseWindow(ByVal hWnd As Long) As Boolean
```

Result:

True if successful.

Parameters:

hWnd — window handle.

Method MoveWindow()

Moves the specified window.

Syntax:

```
MoveWindow(ByVal hWnd As Long, ByVal X As Long, ByVal Y As Long, ByVal
nWidth As Long, ByVal nHeight As Long, ByVal bRepaint As Boolean) As Boolean
```

Result:

True if successful.

Parameters:

hWnd — window handle;
X — new window X coordinate;
Y — new window Y coordinate;
nWidth — new window width;
nHeight — new window height;
bRepaint — True if the window should be redrawn after moving.

Method SendMessage()

Sends a custom message to the specified window.

Syntax:

```
SendMessage(ByVal hWnd As Long, ByVal wMsg As Long, ByVal wParam As Long,
ByVal lParam As Long) As Long
```

Result:

The result corresponds to the result of the Windows API function SendMessage, which is described in MSDN Microsoft.

Parameters:

hWnd — window handle;
wMsg — digital message identifier;
wParam — first parameter of the message;
lParam — second parameter of the message.

Method FindWindow()

Finds a window by its class and title.

Syntax:

```
FindWindow(ByVal lpClassName As String, ByVal lpWindowName As String) As
String
```

Result:

Window handle on success, "0" on failure.

Parameters:

lpClassName — window class;
lpWindowName — window title.

Method FindWindows()

Finds a window by its class and title.

Syntax:

```
FindWindows(ByVal lpClassName As String, ByVal lpWindowName As String, ByVal IgnoreCase As Boolean) As String
```

Result:

Window handle on success or found identifiers separated by commas, empty string on failure.

Parameters:

lpClassName — window class;

lpWindowName — window title;

IgnoreCase — ignore case when searching.

Note:

The "lpClassName" and "lpWindowName" parameters of the method accept the use of wildcards "*" to specify a substring-only search, with any parts of the string ("*") to be ignored when searching.

Method Repaint()

Sends a command to redraw the specified window.

Syntax:

```
Repaint(ByVal hWnd As Long)
```

Parameters:

hWnd — window handle.

Method GetTaskInfo()

Gets the parameters of the process with the specified PID.

Syntax:

```
GetTaskInfo(ByRef pid As Long, Optional ByRef arc As String = "", Optional ByRef app As String = "", Optional ByRef jmpPid As String = "", Optional ByRef wndPid As Long = 0, Optional ByRef wndArc As String = "", Optional ByRef wndApp As String = "", Optional ByVal oldJmpPid As String = "") As String
```

Result:

Returns a string containing the window handle to which the process belongs.

Parameters:

pid — at the input is the identifier of the target process for receiving parameters, at the output is the same process identifier, or the identifier of the process of the call context (if "oldJmpPid" is specified);

The following parameters are optional and are used and specified to return the result:

arc — process architecture;

app — program module name;

jmpPid — hierarchical sequence of process PIDs from the parent process that owns the window to the child target process. The value is returned as a comma-separated value string of PIDs (CSV format);

wndPid — window process handle;

wndArc — window process architecture;

wndApp — window process module name;

The next parameter is optional, its value is obtained from the "jmpPid" parameter in the previous call:

oldJmpPid — hierarchical sequence of PIDs from the window's parent process to its child process in CSV format. This input value is used to determine the process that is the common context of this method's calls.

Method CognateProc()

Returns the closest sibling process.

Syntax:

```
CognateProc(ByVal pid As Long, ByVal moduleName As String, ByVal parentProc As Byte) As String
```

Result:

Related process identifier.

Parameters:

pid — process id;

moduleName — sibling process module name, empty string to ignore;

parentProc — True to get the PID of the closest parent process, False to get the child PID.

Note:

If the "moduleName" parameter is not empty, the closest sibling process is the first process with a module that matches the specified string. The search is case-insensitive, the module name contains the file extension.

Method PidOfWindow()

Gets the process ID of the window.

Syntax:

```
PidOfWindow(ByVal hWnd As Long) As Long
```

Result:

The process ID of the window, "0" for a non-existent window.

Parameters:

hWnd — window handle.

Method WindowsOfPid()

Gets a comma-separated list of window IDs that the specified process has.

Syntax:

```
WindowsOfPid(ByVal pid As Long) As String
```

Result:

CSV list of window IDs, or empty string.

Parameters:

pid — process id.

Method FindChildWindows()

Gets a comma-separated list of child windows that the specified window has.

Syntax:

```
FindChildWindows(ByVal hWnd As Long, ByVal lpClassName As String, ByVal
lpWindowName As String, ByVal IgnoreCase As Boolean) As String
```

Result:

CSV list of window handles, or empty string.

Parameters:

hWnd — window handle.

lpClassName — класс окна;

lpWindowName — заголовок окна;

IgnoreCase — игнорировать регистр букв при поиске.

Note:

The "lpClassName" and "lpWindowName" parameters of the method accept the use of wildcards "*" to specify a substring-only search, with any parts of the string ("*") to be ignored when searching.

Method MonitorInfo()

Gets information on the monitor using its identifier.

Syntax:

```
MonitorInfo(ByVal hMonitor As Long) As String
```

Result:

Gets a JSON-formatted string containing the values of the "cbSize" and "dwFlags" attributes of the MONITORINFO structure (see MSDN).

Parameters:

hMonitor — monitor identifier.

Method MonitorFromPoint()

Gets the ID of the monitor that contains the point with coordinates.

Syntax:

```
MonitorFromPoint(ByVal x As Long, ByVal y As Long) As Long
```

Result:

Monitor identifier.

Parameters:

x — X-coordinate of the point;

y — Y-coordinate of the point.

Method MonitorFromRect()

Gets the ID of the monitor that contains the rectangle.

Syntax:

```
MonitorFromRect(ByVal Left As Long, ByVal Top As Long, ByVal Right As Long,
ByVal Bottom As Long) As Long
```

Result:

Monitor identifier.

Parameters:

Left — X-coordinate of the left edge;

Top — Y-coordinate of the top edge;

Right — X-coordinate of the right edge;
Bottom — Y-coordinate of the bottom edge.

Method MonitorFromWindow()

Gets the ID of the monitor that contains the window.

Syntax:

```
MonitorFromWindow(ByVal hWnd As Long) As Long
```

Result:

Monitor identifier.

Parameters:

hWnd — window handle.

Method GetScreenResolution()

Returns the current monitor resolution to the settings.

Syntax:

```
GetScreenResolution(ByVal monId As Long, ByRef nWidth As Long, ByRef nHeight As Long)
```

Parameters:

monId — monitor identifier;
nWidth — returns the width of the monitor;
nHeight — returns the height of the monitor.

Method NewScreenResolution()

Changes the resolution of the monitor with the identifier to the one specified in the parameters.

Syntax:

```
NewScreenResolution(ByVal monId As Long, ByVal nWidth As Long, ByVal nHeight As Long) As Boolean
```

Result:

True if the screen resolution was changed successfully.

Parameters:

monId — monitor identifier;
nWidth — monitor width for installation;

nHeight — monitor height for installation.

Warning:

The specified width and height values cannot be arbitrary and must be supported by the video card.

Method ScreenShot()

Takes a screenshot and saves it as a JPG graphic file on disk.

Syntax:

```
ScreenShot(ByVal monId As Long, ByVal hWnd As Long, ByVal AFullFileName As String, ByVal AClientArea As Boolean)
```

Parameters:

monId — monitor identifier, "0" for the monitor containing the window or for the primary monitor;

hWnd — window handle or "0". If the value is zero, it takes a screenshot;

AFullFileName — full path and file name to save the snapshot;

AClientArea — take a snapshot only of the client area of the window, valuable if "hWnd" is non-zero.

Note:

If you specify to take a snapshot of a window in a minimized state, the method expands it for the snapshot and then collapses it to its original state.

Method AppBarRect()

Determines the coordinates of the OS taskbar and returns them to output parameters.

Syntax:

```
AppBarRect(ByRef ALeft As Long, ByRef ATop As Long, ByRef ARight As Long, ByRef ABottom As Long) As Boolean
```

Result:

True if successful.

Parameters:

ALeft — X-coordinate of the left edge;

ATop — Y-coordinate of the top edge;

ARight — X-coordinate of the right edge;

ABottom — Y-coordinate of the bottom edge.

Method ScreenClientArea()

Determines the coordinates of the monitor's client area and checks whether the window is within it.

Syntax:

```
ScreenClientArea(ByVal monId As Long, ByVal hWnd As Long, ByRef ALeft As Long,
ByRef ATop As Long, ByRef ARight As Long, ByRef ABottom As Long) As Long
```

Result:

Returns the percentage of the window surface within the monitor's client area, or 100 if the window handle is zero.

Parameters:

monId — monitor identifier;

hWnd — window handle or zero value;

The following parameters are used to obtain the coordinates of the monitor's client area:

ALeft — X-coordinate of the left edge;

ATop — Y-coordinate of the top edge;

ARight — X-coordinate of the right edge;

ABottom — Y-coordinate of the bottom edge.

Method MoveToScrClient()

Moves a window to the client area of the monitor.

Syntax:

```
MoveToScrClient(ByVal monId As Long, ByVal hWnd As Long) As Boolean
```

Result:

True if the move is successful.

Parameters:

monId — monitor identifier;

hWnd — window handle.

Method GetCursorPos()

Gets the coordinates of the mouse cursor.

Syntax:

```
GetCursorPos(ByRef X As Long, ByRef Y As Long) As Boolean
```

Result:

True if successful.

Parameters:

X — X-coordinate of the point;

Y — Y-coordinate of the point.

Method SetCursorPos()

Moves the mouse cursor to the point with coordinates.

Syntax:

```
SetCursorPos(ByRef X As Long, ByRef Y As Long) As Boolean
```

Result:

True if successful.

Parameters:

X — X-coordinate of the point;

Y — Y-coordinate of the point.

Method MouseMoveClick()

Performs a mouse click, if specified in the parameters, it first moves the cursor to the point with coordinates.

Syntax:

```
MouseMoveClick(ByVal nButton As Long, ByVal bDown As Boolean, ByVal bUp As
Boolean, ByVal bMov As Boolean, ByVal AbsX As Long, ByVal AbsY As Long)
```

Parameters:

nButton — mouse button number: "1" is the left button, "2" is the middle button and "3" is the right button;

bDown — when clicked, perform a mouse press (True);

bUp — on click, release the mouse button (True);

bMov — pre-move mouse cursor;

AbsX — X-coordinate of the new cursor position;

AbsY — Y-coordinate of the new cursor position.

Method MouseClick()

Performs a mouse click.

Syntax:

```
MouseClick(ByVal nButton As Long, ByVal bDown As Boolean, ByVal bUp As Boolean)
```

Parameters:

nButton — mouse button number: "1" is the left button, "2" is the middle button and "3" is the right button;

bDown — when clicked, perform a mouse press (True);

bUp — on click, release the mouse button (True).

Method CompareFileBytes()

Compares the data of two files byte by byte.

Syntax:

```
CompareFileBytes(ByVal AFilePathA As String, ByVal AFilePathB As String) As Byte
```

Result:

Returns the numerical result of the comparison, the values can be as follows:

1. "0" — the data of the files match;
2. "1" — files have different data;
3. "2" — the size of one of the files changed while they were being read or a reading error occurred;
4. "3" — the files are of different sizes;
5. "4" — failed to read one of the files;
6. "5" — one of the files was not found.

Parameters:

AFilePathA — name and location of the first comparison file;

AFilePathB — name and location of the second comparison file.

Method GetConsoleText()

Reads console text and returns it as a string.

Syntax:

```
GetConsoleText(ByVal Pid As Long, ByVal lineNo As Long, ByVal lineCount As Long, ByVal lineDelim As String, ByRef AResult As String, Optional ByRef pauseProc As Byte = 7, Optional ByVal AftStr As String = "", Optional ByVal AftSubStr As Boolean = False, Optional ByVal leftTrim As Boolean = False, Optional ByVal widthCWin10 As Integer = 0) As Byte
```

Result:

Returns the numerical result of reading text, values can be as follows:

1. "0" — reading completed successfully;
2. "1" — process with PID has no window or does not exist;
3. "2" — internal error while executing or too slow and timed out;
4. "3" — unexpected messages from injector in target process;
5. "4" — failed to inject library, create overlay window or process has multiple windows;
6. "5" — the console text buffer was locked;
7. "6" — failed to get console output handler;
8. "7" — the process is not a console application;
9. "8" — unknown target process architecture, x86 or x64 supported;
10. "9" — could not read or find the injector file on disk, check permissions;
11. "10" — the injector file was successfully created on disk, please call again to read the result;
12. "11" — the specified string in the console text was not found.

Parameters:

Pid — The process ID of the console window. When launched from a console application, a value of "0" indicates that its console text is being read;

lineNo — line number of the text to be read. The first line number is the line above the console input caret;

lineCount — number of rows to return;

lineDelim — line separator to use when collapsing multiple lines into a single result line;

AResult — the result string where the read text is returned. In case of an error during execution, an error message is returned to it;

The following parameters are optional:

pauseProc — parameter to specify the text reading attempt methods to be used

1. "1" — do not suspend the main window thread;
2. "2" — pause the main window thread to unlock the text buffer and avoid reading artifacts due to current printing to the console window;
3. "3" — read from clipboard by copying.

The summation of parameters 1-3 indicates a combination of these methods, the default value is "7". If this parameter contains a variable, then with an input value of "2" the value "1" will be returned to it in case of successful suspension of the console thread, "0" - in case of failure of this action;

AftStr — if not an empty string, then makes the "lineNo" parameter insignificant and

specifies to search for text after the last occurrence of the specified string. The search for a match is performed without taking into account the case of letters in the strings;

AftSubStr — is valuable only if the "AftStr" parameter is not empty and specifies to search for a substring in console lines rather than an exact match;

leftTrim — if True, then left spaces are discarded in the returned text lines;

widthCWin10 — is applicable only to the new console, which appeared starting with Windows 10. Specifies the width of the text in the console, can vary from "20" to "8192". The default value "80" is the standard width of text lines in the old version of the console. If it contains "0", then it is considered that the length of the lines in the console has an actual value for the current window width. The last value works only with the default font width in the console, which is 8 pixels.

Note:

A special feature of the new console, which appeared with Windows 10, is the removal of line feed characters if the line length during printing is equal to the current width of the console window. That is, in this case, the line is merged with the next line. Therefore, when working with this console, it is recommended to set its width based on the length of lines of 120 characters, and output lines to the console based on the standard line width of 80 characters. The console width of 120 characters corresponds to the size of its default window.

Method EnvironSet()

Reads environment variables defined in the console process and returns them as a string.

Syntax:

```
EnvironSet(ByRef pid As Long, ByRef foundVals As String, Optional ByVal schName
As String = "*", Optional ByVal selMode As Byte = 1) As Byte
```

Result:

Returns the numerical result of reading variables, values can be the following:

1. "0" — reading environment variables was successful;
2. "1" — Wow64 process requests are not supported (x64);
3. "2" — the process is not a console application or is not listed in the process list;
4. "3" — could not find related process with module "cmd.exe";
5. "4" — unknown target process architecture, x86 or x64 supported;
6. "5" — failed to open target process;
7. "6" — failed to request basic process information;
8. "7" — failed to read process PEB;
9. "8" — could not read environment variable data at address;
10. "9" — other runtime error, with Windows OS error ID;

Parameters:

Pid — console process identifier. When launched from a console application, the value "0" indicates that the variables of one of the processes of the current console are being read;

foundVals — environment variables that were read into the target process and matched the

search criteria. If multiple variables were found, they are returned as a single string separated by a newline character (CR, hex code 0x10);

The following parameters are optional:

schName — variable name search string, supports wildcards "*" to specify search only by substrings with parts of the string ("*") that can be ignored during search. If only "*" is specified, all variables with their values will be returned;

selMode — a numeric parameter to specify how to search the console process hierarchy:
1. "0" — search only in the process with the specified PID;
2. "1" — default value, find the latest definition of variables in the latest child process;
3. "2" — find the earliest definition of variables from the console window's parent process.

Method BatchCallStack()

Gets the call stack of files and labels in a process called with the ms-dos "call" command.

Syntax:

```
BatchCallStack(ByVal cmdPid As Long, Optional ByVal AllBatLab As Byte = 0,
Optional ByRef jobReps As String = "", Optional ByRef tabRes As String = "", Optional
ByRef resCode As Integer = 0, Optional ByVal toDosStr As Boolean = False) As String
```

Result:

Returns a call stack string, using the CR character (0x10) as a separator between stack elements, with the first element in the call stack listed first. Each call stack element contains a full parameter list with expanded variable values. File call stack elements contain the full local path to each file; network locations of called files are not supported. On error, returns "FAIL" and the error message in the "jobReps" parameter.

Parameters:

cmdPid — the process ID of the console window in which you want to get the call stack;

The following parameters are optional:

AllBatLab — a numeric parameter to specify the method of selecting elements:
1. "0" — return the full call stack of batch files and labels;
2. "1" — return call stack of batch files only;
3. "2" — return call stack of labels only;

jobReps — returns error messages or an empty string on normal execution;

tabRes — returns a tabulated result in the following form:
1. all arguments of each stack element are enclosed in quotes;
2. all arguments of each stack element are separated by the LF character (0x13);
3. stack elements are separated by the CR symbol (0x10);

resCode — returns an internal error code if an error occurred. "0" on normal execution;

toDosStr — if True, then converts strings with ms-dos control characters to display them correctly in the console.

Notes:

1. Due to internal search limitations, method has the following restriction on the symbols that can be used in the stack: full file names with paths, labels, variable names and their values in the call stack must only have characters in the ASCII range from 0x20 to 0x7E. That is, they can only be English, numeric characters or other characters in the lower half of the ASCII range (see this table for reference);
2. When there are recurrent calls in the call stack, there are ambiguities in the order of its elements. For a workaround, see the library web help for the @callstack macro.

Method SetCodesOfBaseSymbols()

The purpose of the method is to establish the correlation between text symbols and digits of the number system of the specified base (bit depth).

Syntax:

```
SetCodesOfBaseSymbols(ByRef ABase As Integer, Optional ByRef ASyms As String =
"", Optional ByRef ACods As String = "")
```

Parameters:

ABase — a numerical parameter for specifying the base of the number system; if subsequent parameters are defined, its input value is ignored and the base value that was determined from them is returned to it. If the input value is negative, the correlation is performed by generating a random sequence of characters up to the specified number of digits in the number system. Number systems with bases from "2" to "214" are supported. To avoid problems with the representation of text characters in different encodings, it is not recommended to use a base greater than "86";

The following parameters are optional:

ASyms — a string parameter for explicitly specifying the mapping of text symbols to the digits of the number system; if the string is empty, then the generated mapping is returned to it. Using the example of the hexadecimal number system and the standard mapping of text symbols to digits, the input string of this parameter should be "0123456789ABCDEF";

ACods — a string parameter for explicitly specifying the mapping of ASCII hex codes of text symbols to digits of the number system; if the string is empty, the generated mapping is returned to it. Specifying a non-empty input string makes the input values of the previous parameters insignificant. Using the example of the septenary number system with the standard mapping of digits to symbols "0123456", the input string for this parameter is "303132333435".

Note:

Because the internal initialization of the established relationship is carried out in a static attribute, it is preserved throughout the lifetime of the user's COM object. If the parameters "ASyms" and "ACods" are not necessary, they can be omitted for default mapping.

Method DropCodesOfBaseSymbols()

Resets the mapping of text characters to base digits previously specified by the SetCodesOfBaseSymbols() method.

Syntax:

```
DropCodesOfBaseSymbols()
```

Parameters:

absent

Method Radix()

Converts a number as an array of decimal digits from one number system to another.

Syntax:

```
Radix(ByVal ASrcRad As UInt16, ByVal ATgtRad As UInt16, ByRef ANumber() As Integer, Optional ByVal ATgtRank As Integer = -1)
```

Parameters:

ASrcRad — non-negative numerical parameter to indicate the initial digit capacity of the number system;

ATgtRad — non-negative numerical parameter to specify the target base of the number system;

ANumber — integer array, the input must contain a number in the original number system, the output contains a number in the target number system. The number in the array is contained in the form of decimal numerical values of digits. The highest order digits of the number are in the first elements of the array, the lowest order digits are in the last;

ATgtRank — specifies the size of the number after conversion to the target number system. If the specified size is smaller than its size, the highest order digits are discarded, if it is larger, the highest order digits are padded with zeros to the specified size. Any negative value in this parameter will discard insignificant zeros of the highest order digits, the default value is "-1".

Method StrRadix()

Converts a string representation of a number from one number system to another.

Syntax:

```
StrRadix(ByVal ASrcMap As Boolean, ByRef ASrcRad As UInt16, ByRef ASrcSym As String, ByRef ASrcCod As String, ByVal ANumber As String, ByVal ATgtMap As Boolean, ByRef ATgtRad As UInt16, Optional ByRef ATgtSym As String = "", Optional ByRef ATgtCod As String = "", Optional ByVal ATgtRank As Integer = -1) As String
```

Result:

Returns a string representation of a number in the target number system.

Parameters:

ASrcMap — the source string contains a number with a mapping of digits to text characters (True) or a hexadecimal sequence of digits in a number (False);

ASrcRad — non-negative numerical parameter to indicate the initial digit capacity (base) of the number system;

ASrcSym — a string parameter for explicitly specifying the mapping of text symbols to digits in the number system; if the string is empty, then the generated mapping is returned to it (ASrcMap <==> True);

ASrcCod — a string parameter for explicitly specifying the mapping of text character codes to numbers in the number system; if the string is empty, then the generated mapping is returned to it (ASrcMap <==> True);

ANumber — source string with number to convert;

ATgtMap — the target string will contain a number with a mapping of digits to text characters (True) or a hexadecimal sequence of digits in a number (False);

ATgtRad — non-negative numerical parameter to specify the target digit capacity of the number system;

The following parameters are optional:

ATgtSym — string parameter for explicitly specifying the mapping of text symbols to digits in the number system; if the string is empty, then the generated mapping is returned to it (ATgtMap <==> True);

ATgtCod — a string parameter for explicitly specifying the mapping of text character codes to digits in the number system; if the string is empty, then the generated mapping is returned to it (ATgtMap <==> True);

ATgtRank — a numeric parameter to specify the size of the number after conversion to the target number system.

Note:

After completion, method restores the old number base, if it was set earlier. For more information on the parameters "ASrcSym", "ASrcCod", "ATgtSym", "ATgtCod" see method SetCodesOfBaseSymbols(), for more information on "ATgtRank" — Radix(). If the parameters "ASrcSym" and "ASrcCod" are not necessary, empty strings can be specified instead.

Method Shrink()

The main purpose of the method is to compress text, convert the result to a string form for subsequent storage in text, and reverse conversion. The Lempel–Ziv–Welch (LZW) algorithm with subsequent adaptive arithmetic coding is used for data compression. Conversion to text form is carried out by converting the compressed byte array to a number system with a digit capacity that allows numbers to be mapped to text characters. Changing the parameters allows changing its functionality.

Syntax:

```
Shrink(ByVal AShrink As Boolean, ByRef ASrc As String, ByVal AFSrc As Boolean,
ByRef ATgt As String, ByVal AFTgt As Boolean, Optional ByVal AType As Byte = 2,
Optional ByVal AUCase As Boolean = False, Optional ByVal ASplitRank As UInt16 =
16, Optional ByVal AUTF8 As Boolean = False, Optional ByVal AStrRadix As Byte =
84, Optional ByVal AFileRowLen As UInt16 = 5000, Optional ByVal AEcho As
Boolean = False) As Boolean
```

Result:

Returns True if successful.

Parameters:

AShrink — parameter to indicate whether compression is performed (True) or a call to decompression is made (False);

ASrc — a string parameter to specify the original text when compressing, or the compressed data when reversed conversion to text;

AFSrc — indicates that the previous parameter does not contain data, but the name of a file with data (True);

ATgt — a string parameter to return compressed data during direct conversion, or the original text during decompression;

AFTgt — indicates that the previous parameter will contain not data, but the name of the file with data (True);

The following parameters are optional and their default values are optimal:

AType — a numeric parameter to specify the type of conversion, can have the following values:
- "0" <=> simple LZW transformation, can only be used for compression and storage in binary file;
- "1" <=> LZW transformation with arithmetic coding for compression into binary file too;
- "2" <=> default value. LZW conversion with adaptive arithmetic coding, the result is converted to a string representation of numbers in the specified number system, using the selected mapping of digits to text characters;
- "3" <=> LZW transform with coding, the result is a hexadecimal string representation of the bytes of the compressed array as is;

AUCase — indicates that the source text may be converted to uppercase (True), defaults to False. Only valuable when calling the method for compression;

ASplitRank — defines the length of the byte subarrays in the resulting compressed array that will be converted to strings (numbers) of the specified number system, by default "16". The minimum value is "8", and due to limitations in conversion performance it is not recommended to choose a value greater than "32";

AUTF8 — indicates that the source text file may be a Unicode file with a service character sequence with codes "&#EF;&#BB;&#BF" at its beginning. Is significant only when compressed, defaults to False;

AStrRadix — digit capacity of the number system for converting compressed data to text, default value "84". The value "86" is the maximum "reliable" value with the same representation of characters in different encodings. The parameter is valuable only if "AType" is "2";

AFileRowLen — length of lines when writing compressed data to a file as text, defaults to "5000". Only valuable if "AType" is "2";

AEcho — specifies that the result should be printed to the screen (True) rather than returned to "ATgt", defaults to False. Only meaningful if "AFTgt" is False.

Note:

If the "AType" parameter is equal to "2" and you intend to use the built-in mapping of text characters to numerals in the number system, then method does not require a preliminary call to SetCodesOfBaseSymbols().

Method GetDosString()

Performs simple string transformations to correctly display control characters in the ms-dos console.

Syntax:

```
GetDosString(ByVal ASrcString As String, AForEcho As Boolean) As String
```

Result:

Returns a string with the necessary transformations.

Parameters:

ASrcString — source string;

AForEcho — if True, will print the converted string to the screen instead of returning it (False).