Macro Toolworks

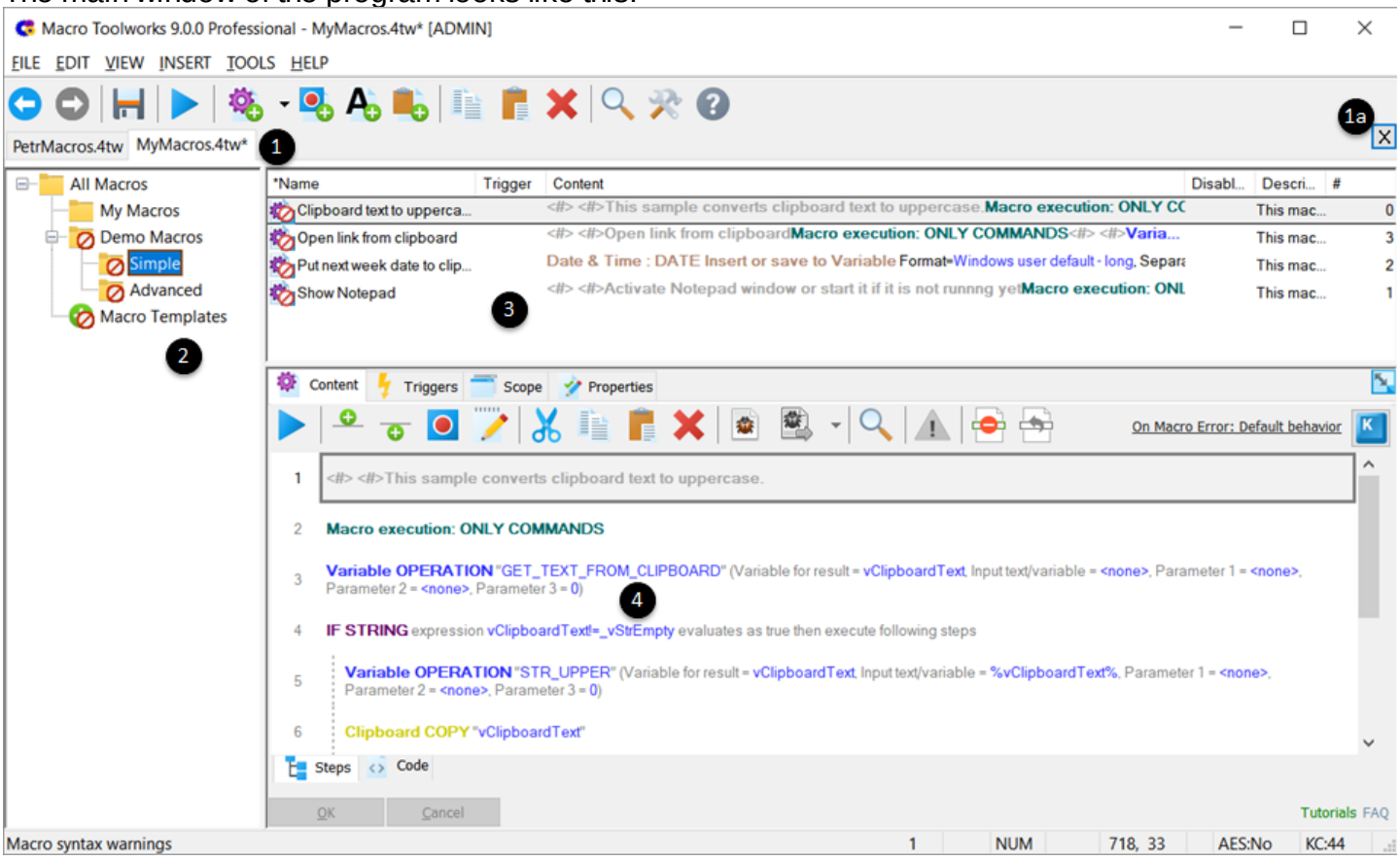
Table of Contents

Overview

Macro Toolworks 9.3.0, www.macrotoolworks.com

The Macro Toolworks allows user to create "macros" to simplify, speed up, and remove errors in repetitive tasks (such as typing the same phrases, copying/deleting files, downloading files, etc.). The Macro Toolworks allows user to create and use at once multiple macro files that are represented as tabs in the main window (see below). Macros in each macro file are organized in macro groups that have a tree structure. Each macro group can contain multiple macros. Macros from the selected group are showing in the list. The macro selected in the list is displayed in the macro editing are (bottom-right area of the main window) where user can edit its content and properties.

The main window of the program looks like this:



Legend:

1 - Tabs that represent macro files open.
1a - Click to close macro file represented by currently selected tab.
2 - Tree structure of macro groups.
3 - List of macros from the currently select macro group.
4 - Macro editing area.

After the Macro Toolworks is installed, the program group is created in the Start menu. To start the Macro Toolworks just click it's item in the Start menu. There is a program icon showing in the Task bar tray area when program is running. Clicking on the icon will show/hide the program's main window (as shown on the figure above), right-click on the tray icon shows a menu with basic options. When the program is exited (from tray icon menu or when shutting down Windows) it remembers if the main window was showing or hidden and starts it in the same state next time.

Support, Feedback, Privacy, Uninstall

Support

Please use one of the following options when reaching a support:

- Support Page
- E-mail support

Feedback

Please provide us feedback so that we can improve the software.

- E-mail support

Privacy

- The software does not collect and send out any information (the software is only communicating with Pitrinec Software (https://www.pitrinec.com) home page in order to determine if a new version is available - this feature can be turned off in Program Settings dialog box).
- Pitrinec Software does not share or provide to others any information that is received from users during support communication.
- Pitrinec Software does not share or provide to others any user identification information such as names or e-mails.

Uninstall

Use standard Windows means (Control Panel in Windows 7, Settings Apps & Features in Windows 10) to uninstall the program if needed.

This Help Document Limitations

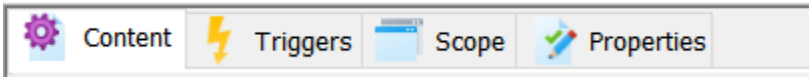
- The Macro Toolworks is distributed in multiple editions (Free, Professional). It is possible that features described in this help document are not available in all the product editions.
- This help documentation contains the product screen-shots. Since the product is available for multiple versions of Windows, and since the product evolves dynamically it can happen that screen shots from this document do not fully match the latest users experience.
- This document only describes features that are not obvious.

Macro

The term "macro" has many meanings in different contexts. In the context of the Macro Toolworks the macro is a user defined set of data that when executed by Macro Toolworks automatically does a task that user is typically doing often manually. There are some examples of macros:

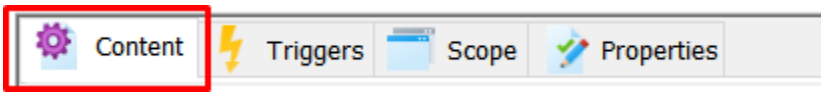
- A macro inserts larger (several paragraphs) predefined text to an e-mail client, document editor or chat client when user hits a hot-key or types a text shortcut.
- A macro copies Excel document files modified during last week, put's all of them to password protected zip file and send's it to a remote server. Such macro can be scheduled to run every Friday night.
- A macro displays a form to be filled by user, and based on the data inserted the macro does some more complex task (for example: updates Excel sheet and sends it out via e-mail).
- A macro downloads a web page, detects a keyword, and sends notification e-mail.

Macro can be edit/modified in macro editing area that consists of four tabs:



- Content
  The content defines what macro does when it is executed. There three types of macros: text macro, clipboard macro, and general macro.
- Triggers
  The triggers define how user starts the macro execution (hot-key, time scheduled, etc.).
- Scope
  The scope defines in what applications (windows) the triggers work - if the macro should run in all applications or just in specific one.
- Properties
  In this tab it is possible to set the macro name, icon, and several other properties.

Content



The content defines what macro does when it is executed. There are three types of macros the Macro Toolworks allows you to create:

- **Text Macro**
  The content of this type of macro is rich text (unicode, rich text with formatting, images, and links). When macro is executed it inserts the text to other application such as e-mail client, word processor, etc.

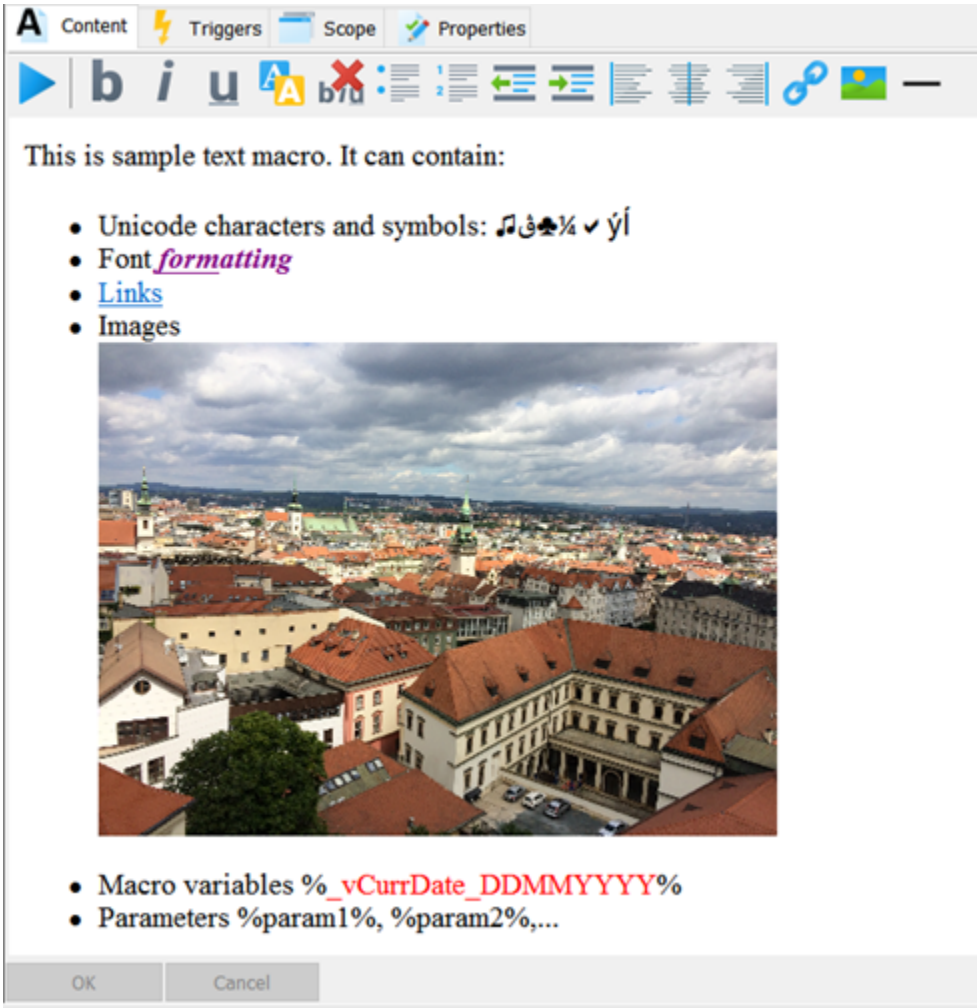
- **Clipboard Macro**
  The content of this macro is data (graphics, tables, text, file links, etc.) saved (persisted) from clipboard. When macro is executed it pastes the clipboard data to other application.

- **General Macro**
  This type of macro can consist of various macro commands to manipulate keyboard, mouse, files, folders, clipboard, ftp, web, etc. This type of macro can execute other macros including text macros or clipboard macros. General macro can be either created manually or it can be also recorded.

Text Macro

Text macro allows user to create a rich unicode text with full font formatting and images. The macro can be created by clicking on the "Add Text Macro" button as shown here:

The macro can look like this:



These text formatting features are supported:

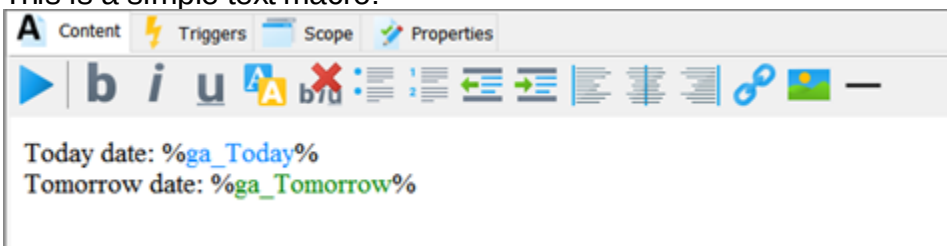
- Bold
- Italic
- Underline
- Fonts
- Colors
- Bullets
- Numbering
- Indention
- Alignment
- Hyper-links
- Pictures
- Horizontal lines

When macro is executed then the macro text is copied to the clipboard and pasted in other application.

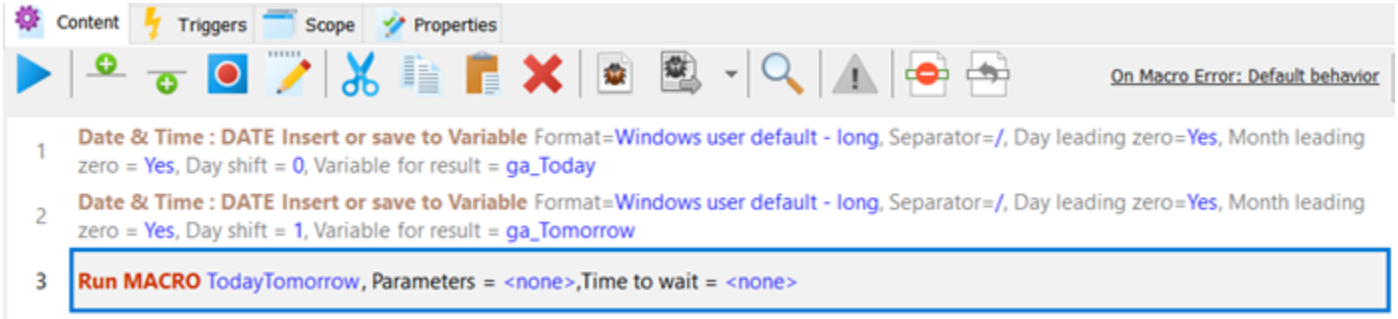
Variables Replacement

Text macro can also contain variables (see variables and system variables). In such case, when macro is executed then the variables are replaced by the variable value - for example by the current date. In order to use variable replacement feature the text macro must be run from within a general macro using RUN MACRO / <run> command. The general macro sets the variables to the required values and then the text macro is executed using RUN MACRO / <run> command. Here is an example:

This is a simple text macro:



And this is the general macro that sets "ga_Today" and "ga_Tomorrow" variables and then runs the text macro:



(Note: Make sure that user defined variables used in text macro always have "ga_" prefix - for example, "ga_MyVariable1".)

When the general macro runs the result is this (in MS Word):

Today date: Sunday, March 10, 2019
Tomorrow date: Monday, March 11, 2019

Expression Replacement

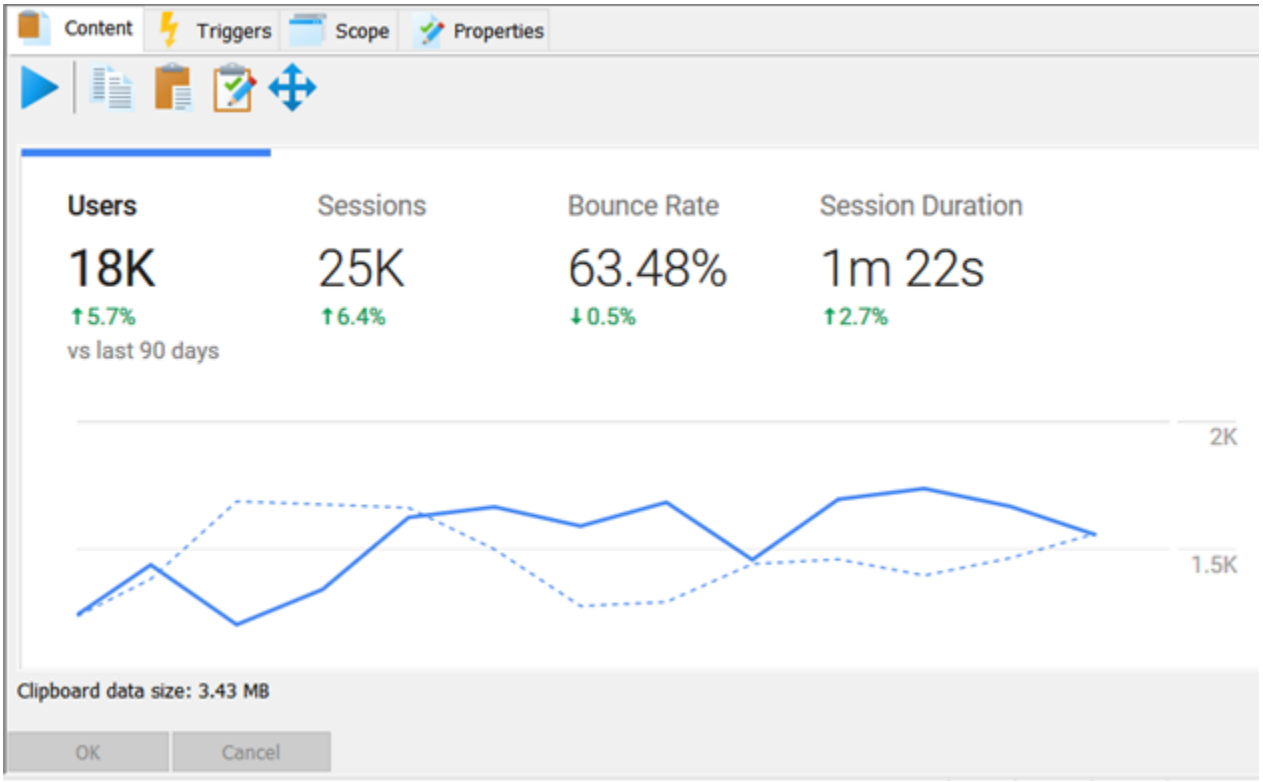
Text macro can also contain expressions (see expressions). For example, the text "*1+1 = EXPR(1+1)*" will provide output "*1+1 = 2*".

Clipboard Macro

Clipboard macro allows user to persist any clipboard content within macro file and paste this clipboard content any time. The clipboard macro can be created by clicking on the "Add Clipboard Macro" button:

(Note: When creating clipboard macro the current clipboard content is immediately used.)

The macro can look like this:



When the macro is executed then the persisted macro content is loaded to the clipboard and then pasted to the application in which the macro is executed.

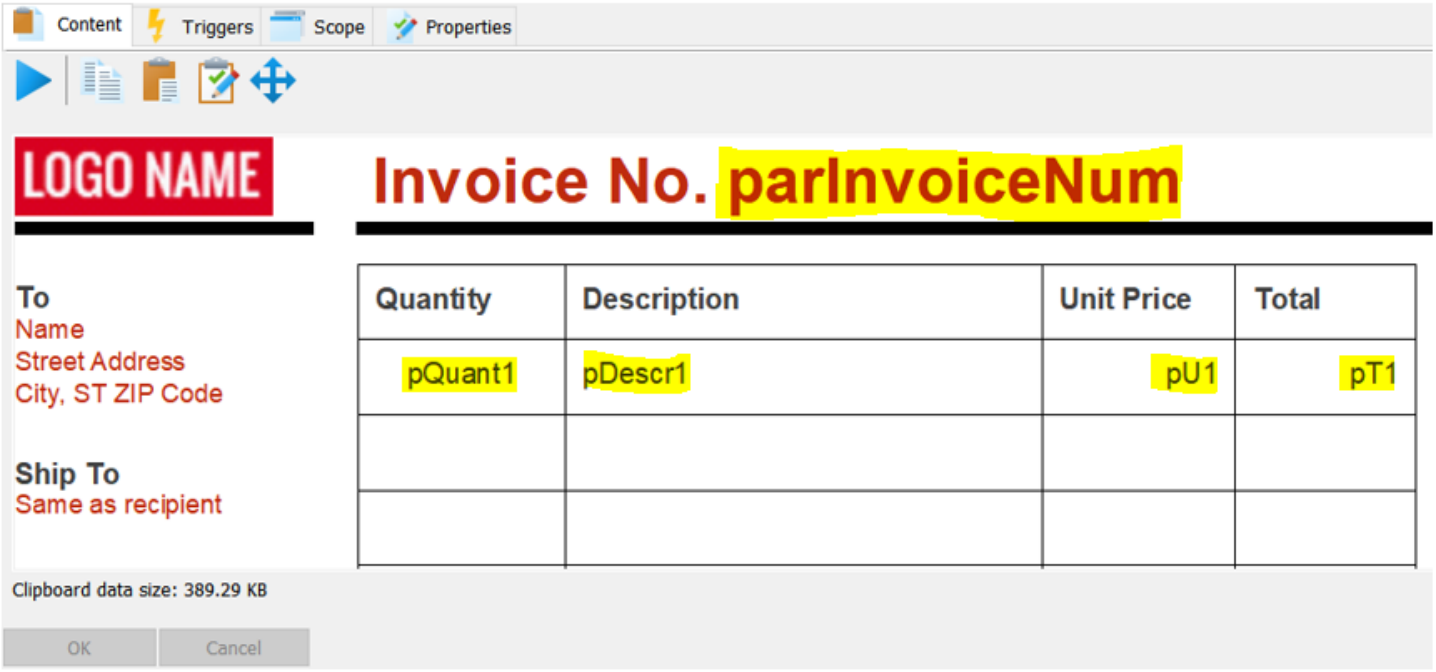
There are these operations supported from the clipboard macro toolbar:

- Execute macro
- Copy the clipboard macro content to clipboard
- Paste the clipboard content to the clipboard macro
- Remove unwanted clipboard formats from the macro
- Shrink the clipboard data preview to fit the view area

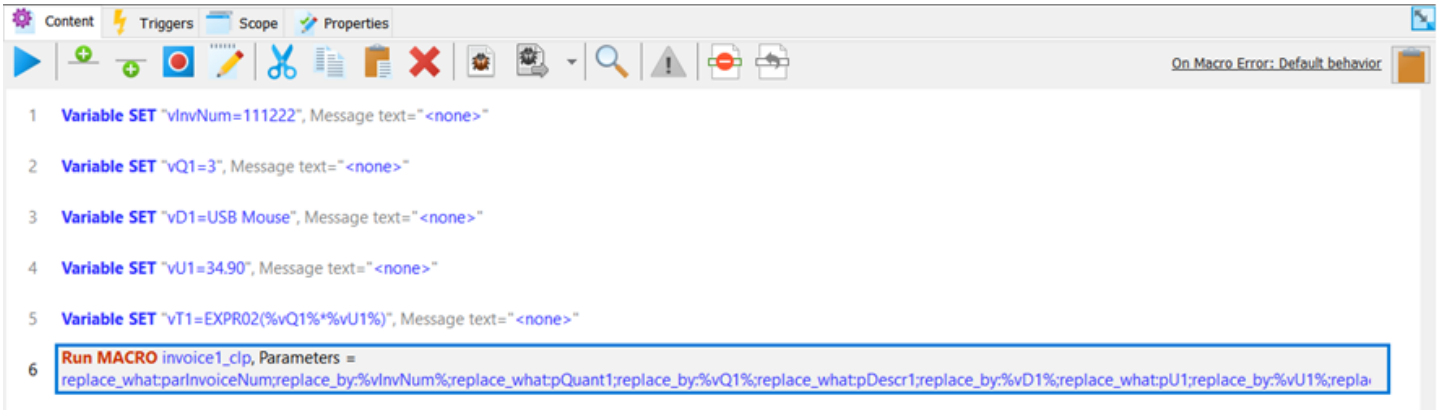
Parameters Replacement

Clipboard macro can also contain parameters to be replaced. In such case, when macro is executed then the parameters are replaced by the required values. In order to use parameter replacement feature the clipboard macro must be run from within a general macro using RUN MACRO / <run> command. Here is an example:

This is the Clipboard Macro (named "invoice1_clp"). It contains an invoice template created in MS Word. Notice highlighted parameters:



This is the general macro. It defines variables used to replace the parameters in the clipboard macro:



When this general macro is run then it produces output like this (in MS Word). Notice the parameters replaced by defined values:

LOGO NAME

Invoice No. 111222

To

Name
Street Address
City, ST ZIP Code

Ship To

Same as recipient

Instructions

Delivery Instructions

Quantity	Description	Unit Price	Total
3	USB Mouse	34.90	104.70

Variables Replacement

Clipboard macro can also contain variable in format **%variable%**. If the variable exists (see variables and system variables) then it is replaced by its value. This provides the same benefit as parameters replacement but it is in many cases easier.

Expression Replacement

Clipboard macro can also contain expressions (see expressions). For example, if the clipboard macro contains this text "*1+1 = EXPR(1+1)*" then it will provide output "*1+1 = 2*".

General Macro

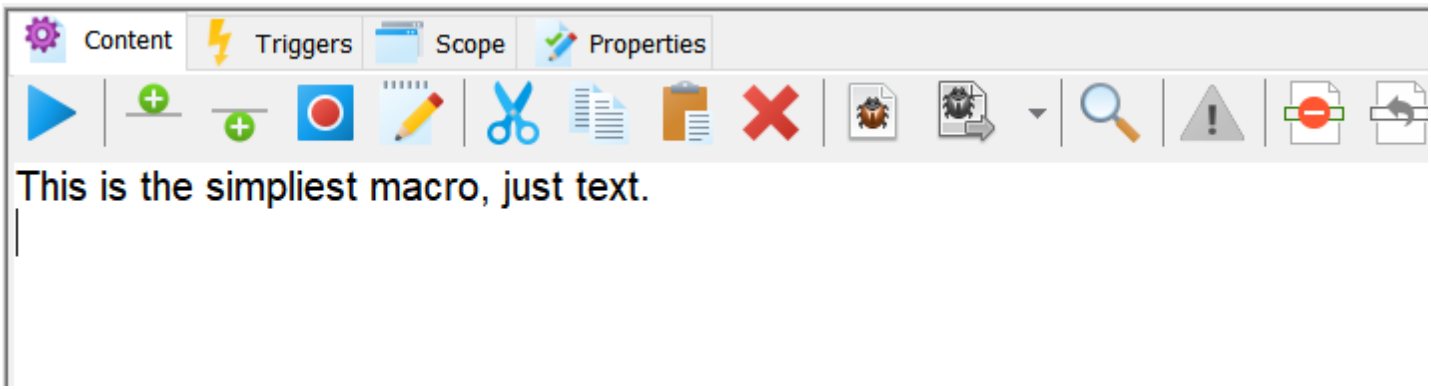
General macro allows user to create a macro that:

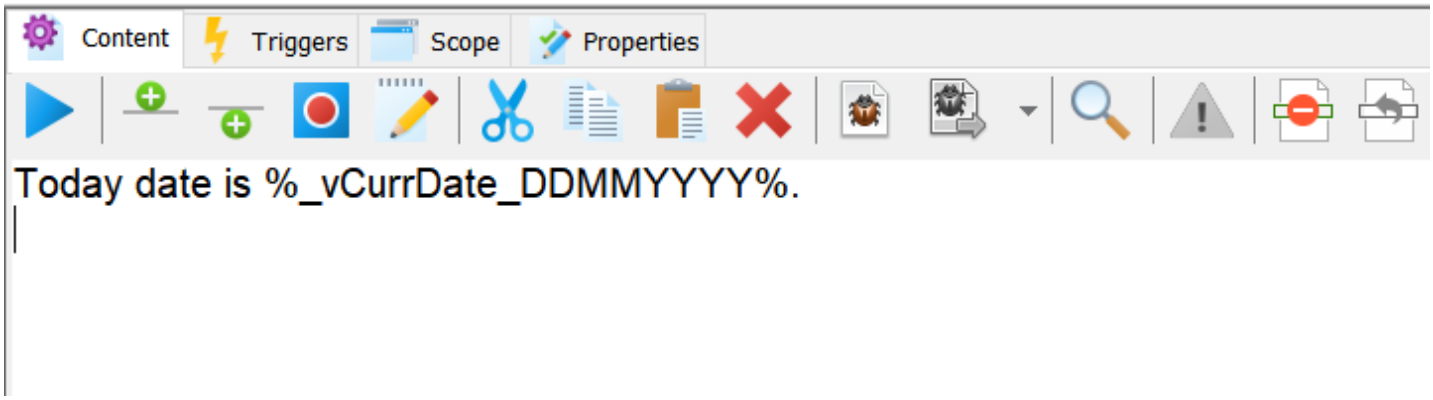
- **Inserts a plain text** (similar to text macro but just plain text, no rich content)
  If macro content only contains a text like "Hello, world!" and this macro is executed then all what happens is that the text "Hello, world!" is inserted to the application in what the macro was started. It can be configured if the text is inserted through clipboard (copy text to clipboard and then paste it in the application) or as a sequence of keystrokes (mimic keyboard).

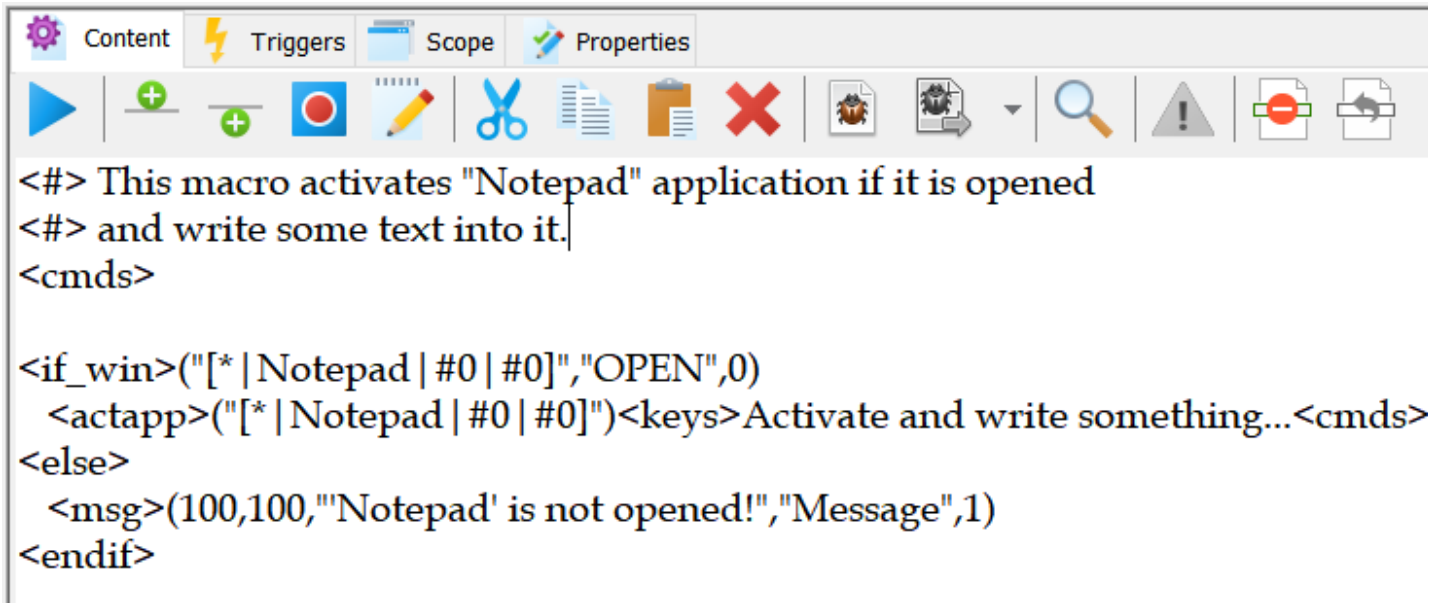
- And/or **executes various** macro commands to manipulate mouse, keyboard, files, windows, clipboard, variables, registry, etc. It is typical that macros consist from both plain text to insert and macro commands. For example, the macro can look like this:
  "Hello, <wx>(1000)world!"
  Such macro when executed inserts text "Hello, ", then it waits for one second and then it inserts text "world!".

There are some examples:

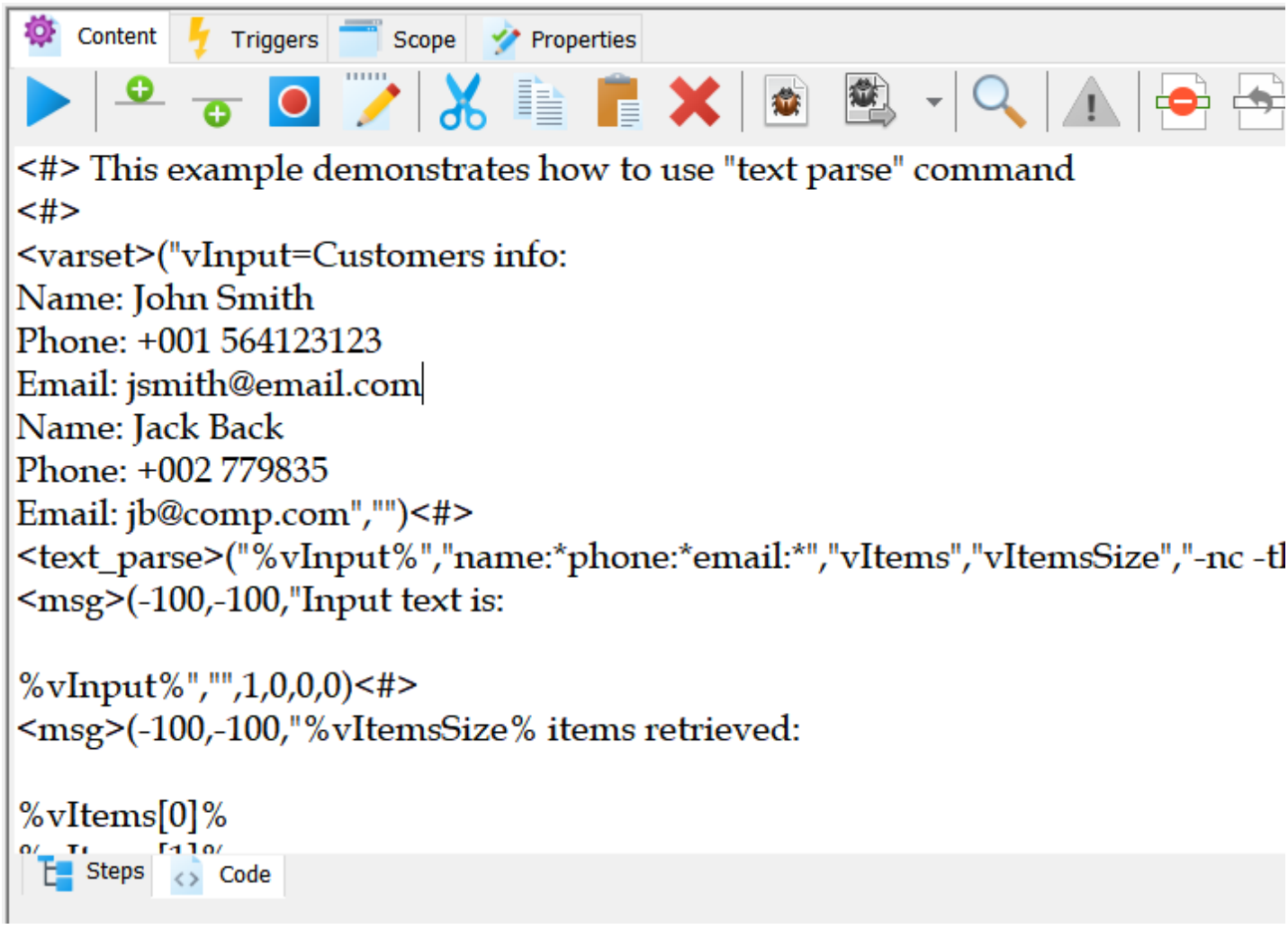
Sample macro



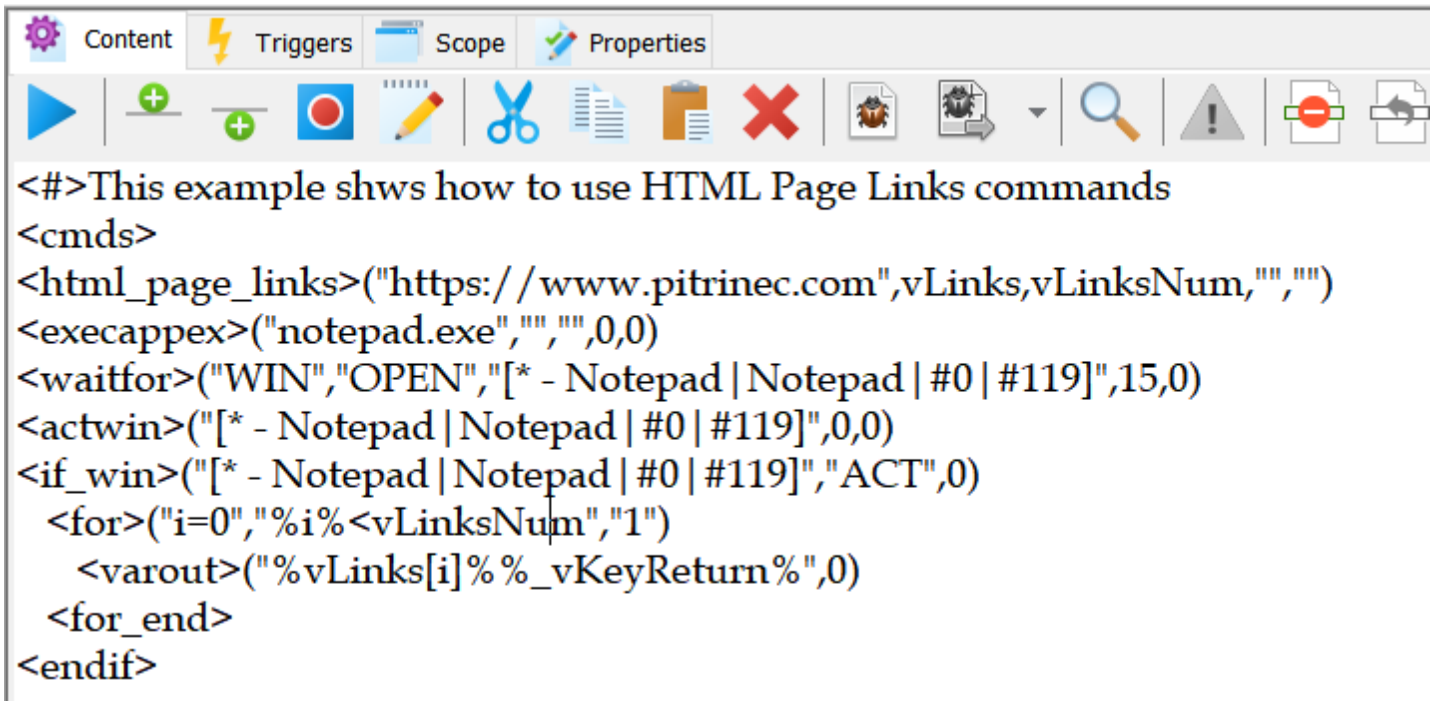





```
<#> This macro activates "Notepad" application if it is opened
<#> and write some text into it.
<cmds>

<if_win>("[*|Notepad|#0|#0]","OPEN",0)
  <actapp>("[*|Notepad|#0|#0]")<keys>Activate and write something...<cmds>
<else>
  <msg>(100,100,"'Notepad' is not opened!","Message",1)
<endif>
```




```
<#> This example demonstrates how to use "text parse" command
<#>
<varset>("vInput=Customers info:
Name: John Smith
Phone: +001 564123123
Email: jsmith@email.com
Name: Jack Back
Phone: +002 779835
Email: jb@comp.com","")<#>
<text_parse>("%vInput%","name:*phone:*email:*","vItems","vItemsSize","-nc -tl
<msg>(-100,-100,"Input text is:

%vInput%","",1,0,0,0)<#>
<msg>(-100,-100,"%vItemsSize% items retrieved:

%vItems[0]%
```

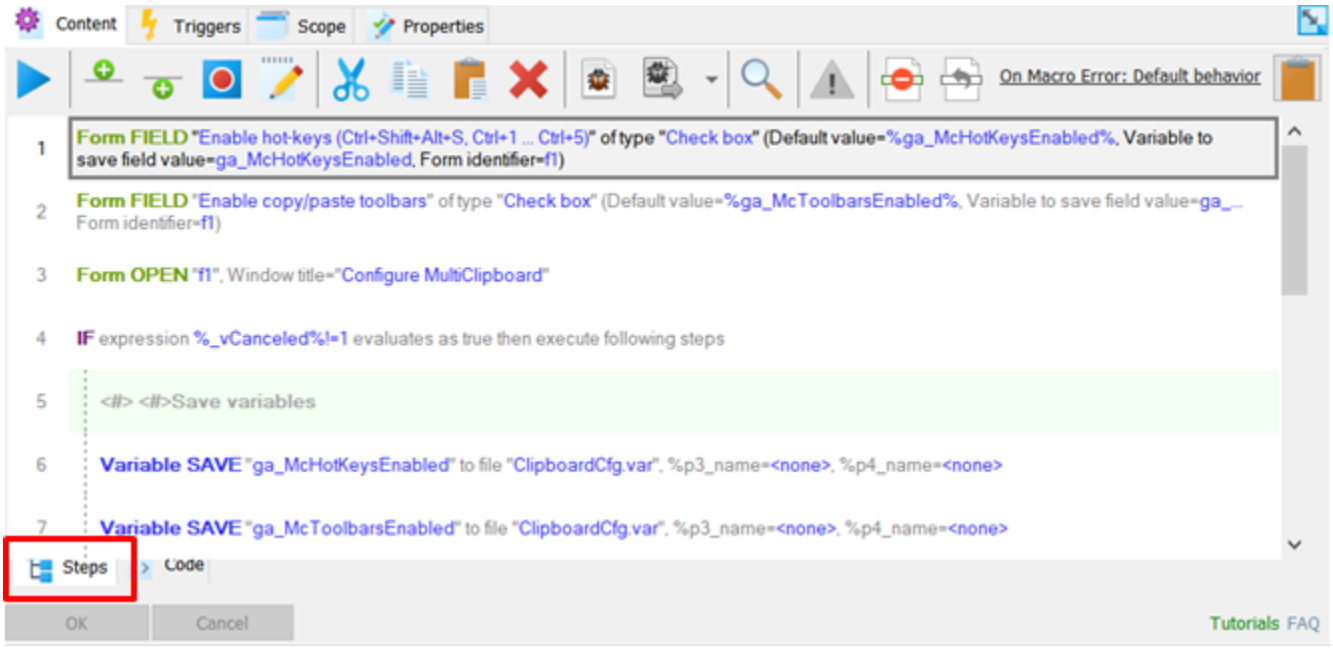


The macro can be created by clicking on the "Add General Macro" button as shown here:

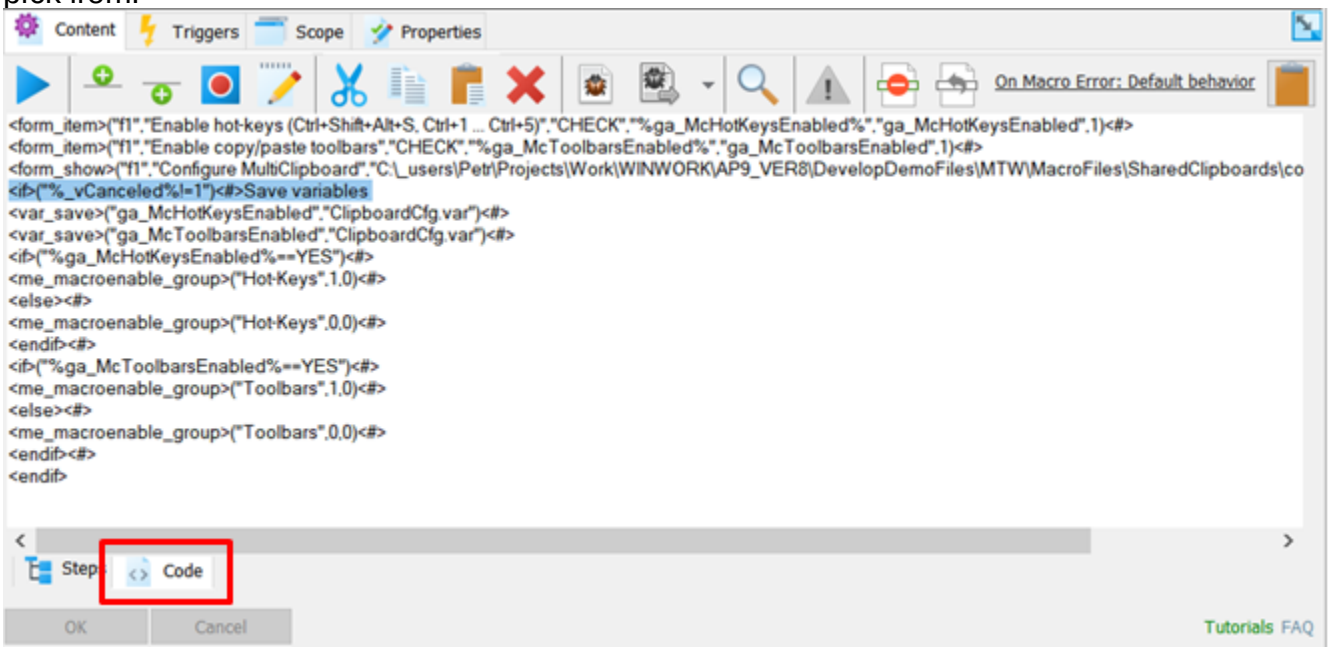
Steps vs. Code

It is possible to view and edit macro in two modes: **Steps** and **Code**. Does not matter what editing mode user uses, the macro always does the same thing. It is possible to switch between these two modes any time using a tab shown on the figure below.

- **Steps**
  If "Steps" is selected then the macro is displayed as a formatted sequence of steps. The macro steps can be added, deleted, copy/paste, dragged and dropped....

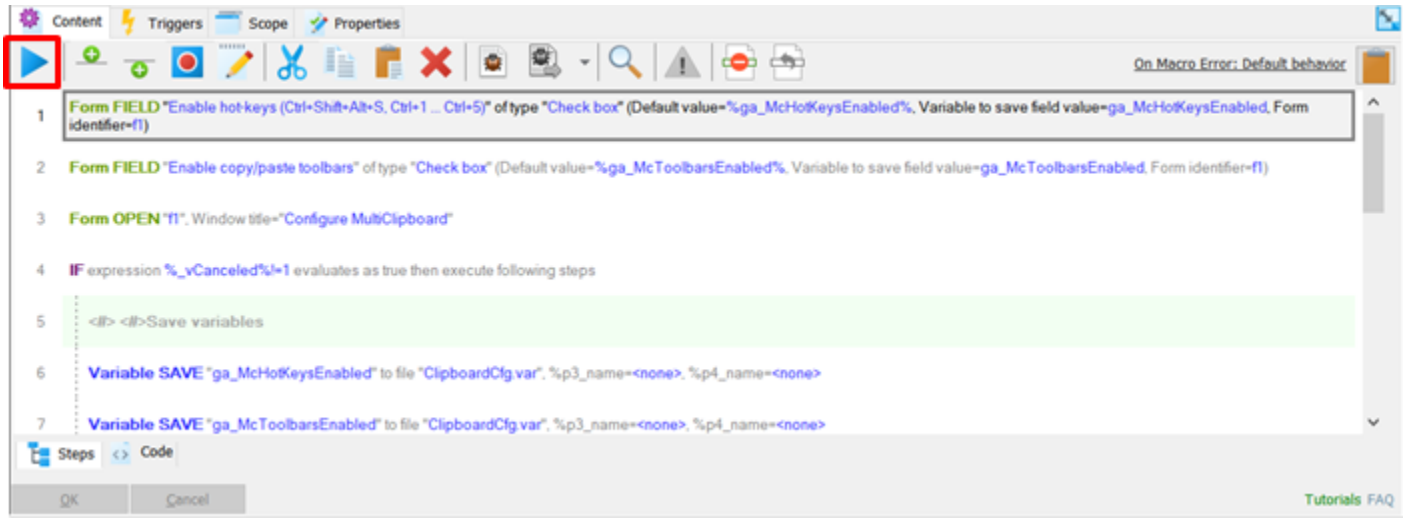


- **Code**
  If "Code" is selected then a plain macro text with its specific commands syntax is displayed. In this case the macro can be edited as a text. It is not needed to know the macro commands syntax to add a new command - when < is typed then a list of available commands is showed to pick from.



Run Macro

One of the way how to execute macro is by clicking on "Run Macro" button in the toolbar:



It is possible to run just selected steps of the macro. This can help with development of bigger macros. To run just a subset of macro steps do this:

1. Select steps (click on them holding down "Ctrl" key). When the steps are selected then release the "Ctrl" key.
2. Then press "Alt" key and hold it pressed.
3. Click on "Run Macro" icon.

Adding Command

Commands can be added to the macro by clicking on "Add Command" toolbar button or by hitting "+" key on numeric pad.

If in "Code" mode, of course, the macro is edited as a plain text and macro can be typed the same as any other text.

Macro Recording

Macro recording can be started by clicking on "Record Macro" button:

See more about macro recording.

Command Editing

A command editor can be open by clicking on "Edit Command" toolbar button or by double-click on the command.

Macro debugging

Macro can be executed step-by-step and content of the variables can be inspected during the debugging. Click on "Start debugging" to enable other debugging operations:

- **Go to debug break**

There is "Debug BREAK POINT" (<-dbp->) macro command that can be put in macro as a point where to postpone debugging. This helps to quickly get to certain step in the macro instead of getting there step by step.

- **Go to cursor position**
  During debugging it is possible to navigate to other macro step/command and select it (move cursor and click). This operation then runs the macro until it reaches the selected step/command.

- **Show command to execute**
  If during the debugging user scrolls to other step/command and selects it then this operation jumps back to the macro step/command that will be executed next.

- **Show variable content**
  This operation displays a window that shows macro variables current value.

- **Stop debugging**
  Terminates debugging.

Find

It is possible to search in macro for a text occurrence:

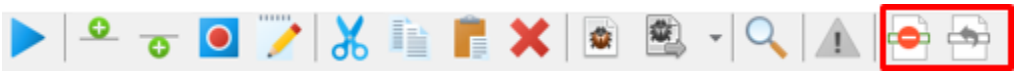
The Macro Toolworks searches in the text that is displayed based on **Steps** or **Code** mode. This means that for example if user searches for the <run> command occurrence while having Steps mode selected nothing will be found (because <run> command is in Steps mode displayed as more descriptive "Run MACRO").

Macro validation

Macro text is automatically validated. If a suspicious syntax is detected then a warning icon is highlighted and enabled. Warnings detected can be displayed by clicking on the icon.

Comment Out

It is possible to comment out a selected part of the macro (one or more commands). The comment out part of the macro is not executed. It is possible to uncomment the block again - just select portion of the commented block and click toolbar button.



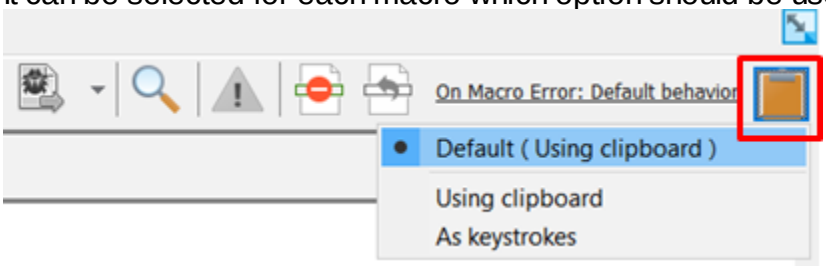
Using Clipboard or As Keystrokes

As it was said, the general macro inserts plain text (such as "Hello, World!"). There are two ways how the text can be inserted:

1. Using Clipboard
   The macro text is copied (similar to Ctrl+C) to the Windows clipboard and then pasted (similar to Ctrl+V) in the other application.

2. As keystrokes
   The macro text is sent to other application as a stream of keystrokes the same as when user types it on keyboard.

It can be selected for each macro which option should be used:



If the "Default" option is selected then the option is the one that is defined in the program settings.

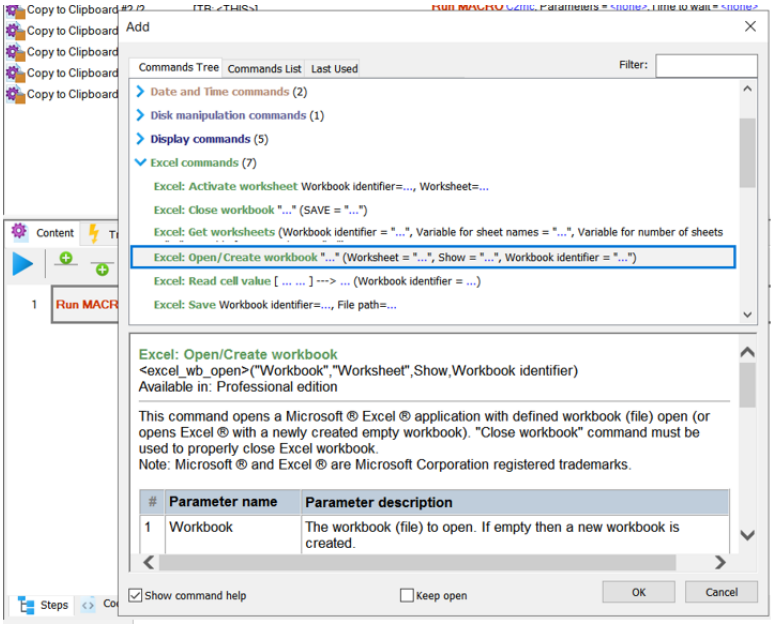
On Error

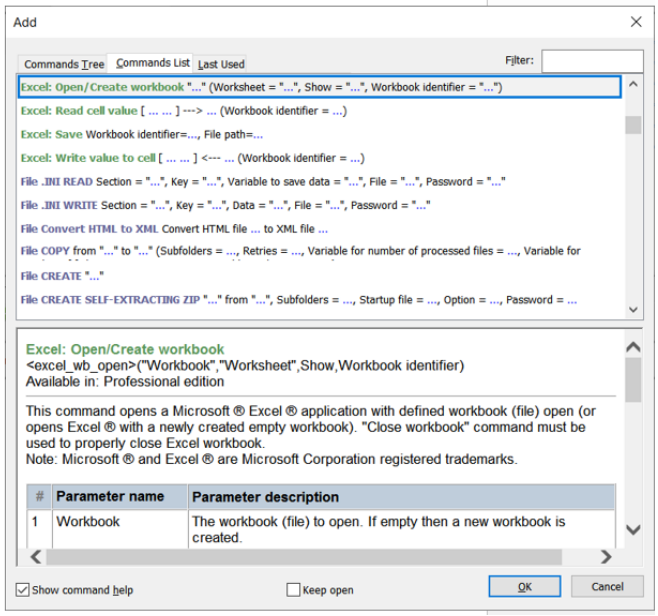
It is possible to define what should happen when a macro command fails:

Read here more.

Add Macro Command

Macros can contain commands that can add variety functionality to the macro. Macro commands can be added in Macro Steps / Macro Text view by hitting + key on numeric keyboard or by clicking on  buttons.
Available commands are listed in the alphabetical order in the window shown below. The window either displays the macro command names (Macro Steps, figure on the left) or macro commands syntax (Macro Text, figure on the right) depending if Macro Steps or Macro Text tab is active in the Macro Steps / Macro Text view.





When a command is selected then its help content is shown in the lower area of the window. The help content contains also a simple example that helps to understand how to use the command.

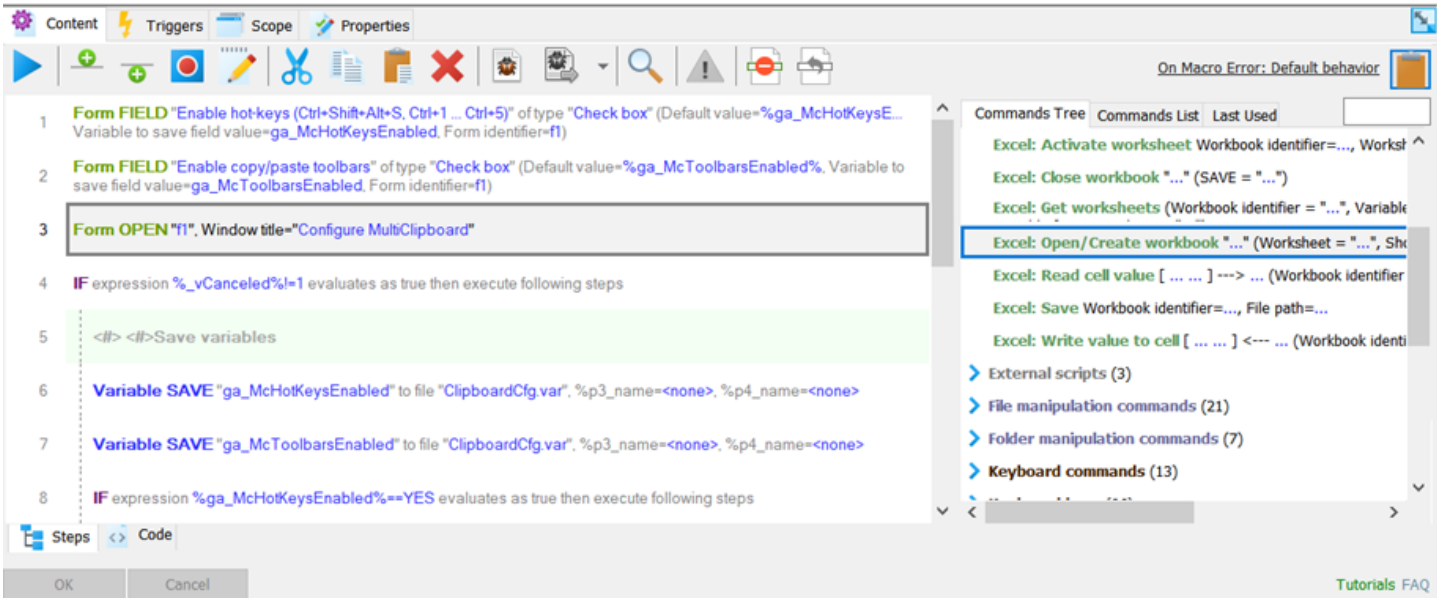
There are three tabs that allows to navigate in the available commands different ways:

- **Commands List**
  All the commands are alphabetically listed.

- **Commands Tree**
  The command categories are listed. Double-click on a category to get the list of commands available in the given category. Go back to the category list by double clicking on the first line with "..".

- **Last Used**
  Number of last used commands are listed in this tab. Use this tab to quickly access commands you are adding repeatedly.

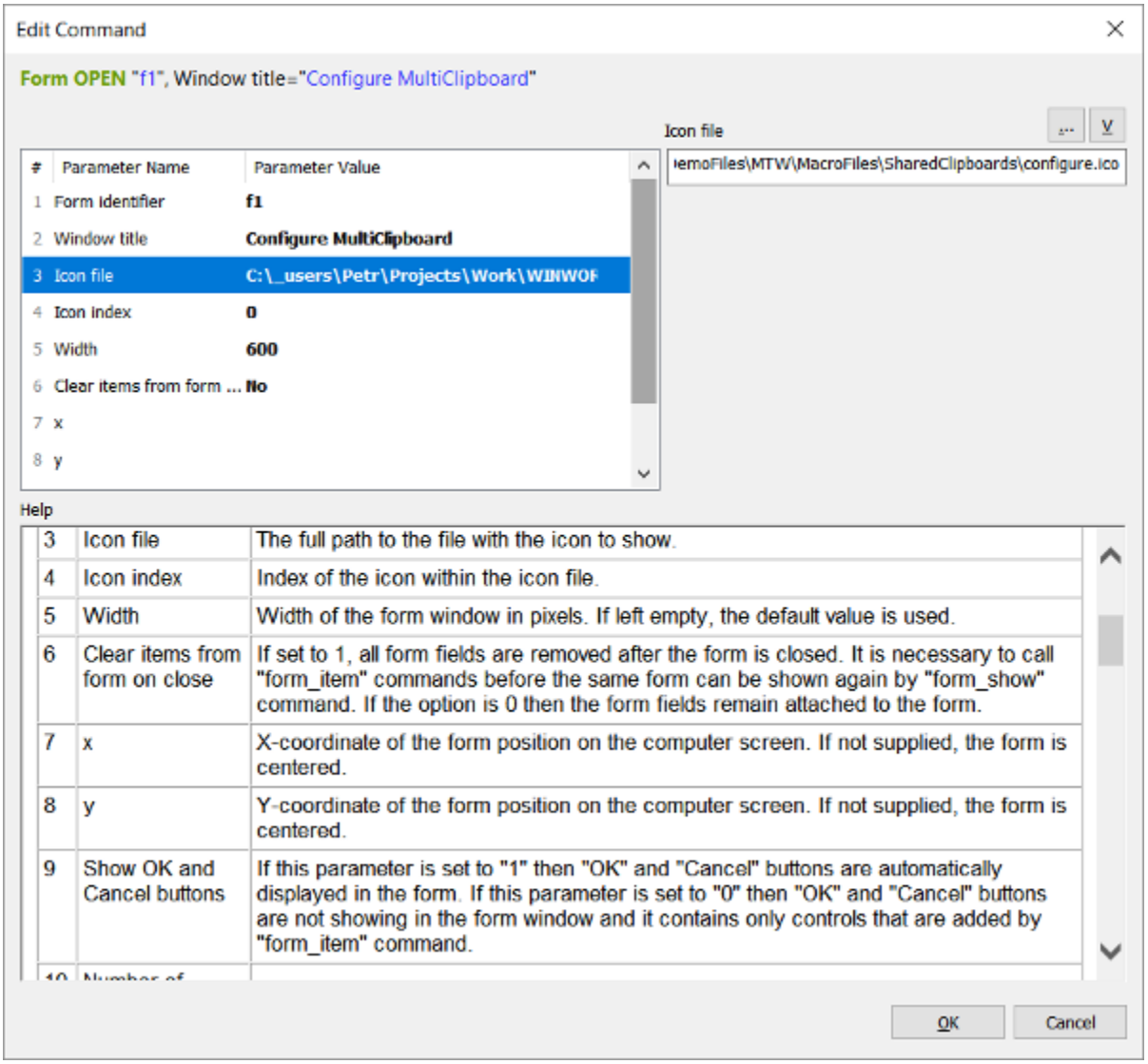
There is an option to keep the "Add" window open within the macro editor. Just click on "Keep open" option to get the window open like this:



Just double-click on the command to get it it added to macro.

Macro Command Editor

Macro commands can be edited in by double-clicking (or Enter key) on the given command or by clicking on the  toolbar button.



All the command parameters are listed in the upper-left area of the window. The selected parameter can be edited (modified) in the upper-right area. Notice the little buttons "..." and "V". The "..." button is available just for certain type of parameters (such as files, folders, macro names, etc.) and it opens a browse window. The "V" button open a window with variables and system variables and it allows to insert variable as a parameter. It is possible to navigate from one parameter to other parameter using "Ctrl+Up" or "Ctrl+Down" keys.
Help is displayed for each parameter in the bottom area of the window.

On Macro Error

It is possible to define what should happen when a macro command fails. There are these options:

- Default behavior
  If this option is selected then the behavior is defined by settings in the (parent) macro group.
- Show error message
  A message window with error description is shown on the screen.
- Log error and stop macro
- Log error and continue
- Run other macro
  Other macro previously selected by user is run. The macro takes as the parameter the error description.

Recorded Macro

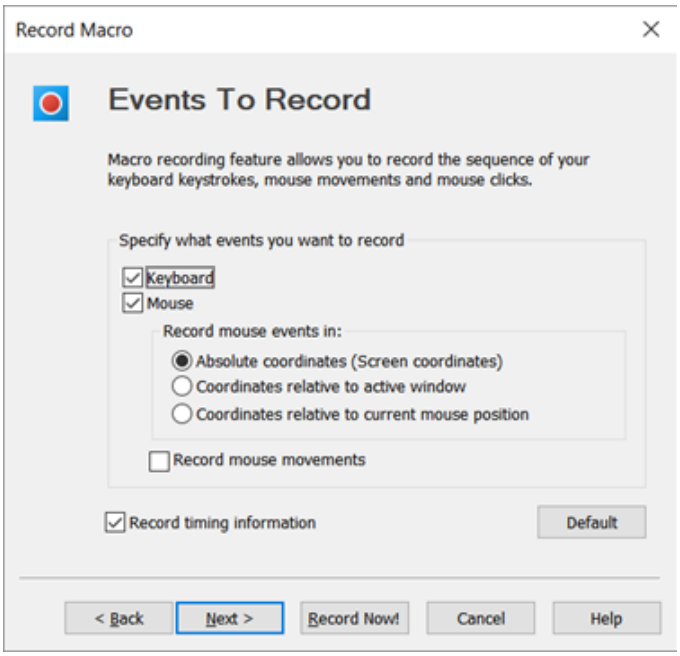
One option how to create a new macro is record what user is doing with keyboard and mouse (typing on clipboard, clicking with the mouse, etc.). The recorded macro can then reply it back any time later. The recorded macro is created automatically during the macro recording. It is possible to later manually edit and tweak the macro if needed.

Click "Record macro" button in the tool bar.

1. Follow instructions in macro recording wizard.
2. After recording is finished, edit macro content if needed.
3. Assign and configure one or multiple macro triggers.
4. Configure macro scope if needed.
5. Assign macro name and configure other macro properties.

Events to Record

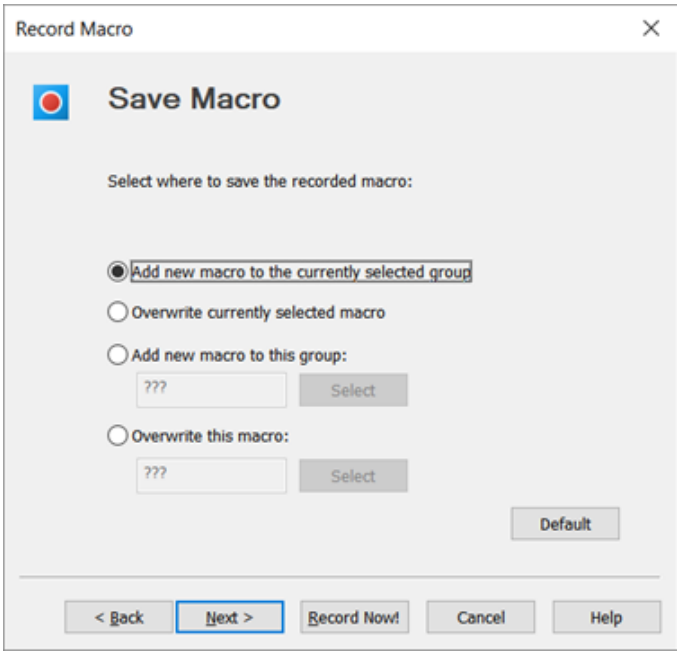
In this first step, the user can select what events (mouse or keyboard or both) should be recorded:



- **Keyboard** - if checked, all keyboard events (keystrokes) are recorded.
- **Mouse** - if checked, all mouse clicks are recorded.
- **Record mouse movements** - if checked, mouse movements are also recorded so that mouse cursor movements are smooth when playing macro back. (Note: The size of macro recorded is much bigger.)
- **Mouse coordinates** measures in:
  1. Absolute (screen) coordinates
  2. Coordinates relative to active window
  3. Coordinates relative to current mouse position
- **Record timing data** - if checked, timing data is saved during macro recording.

Use "Default" button to select default settings.

Save Macro

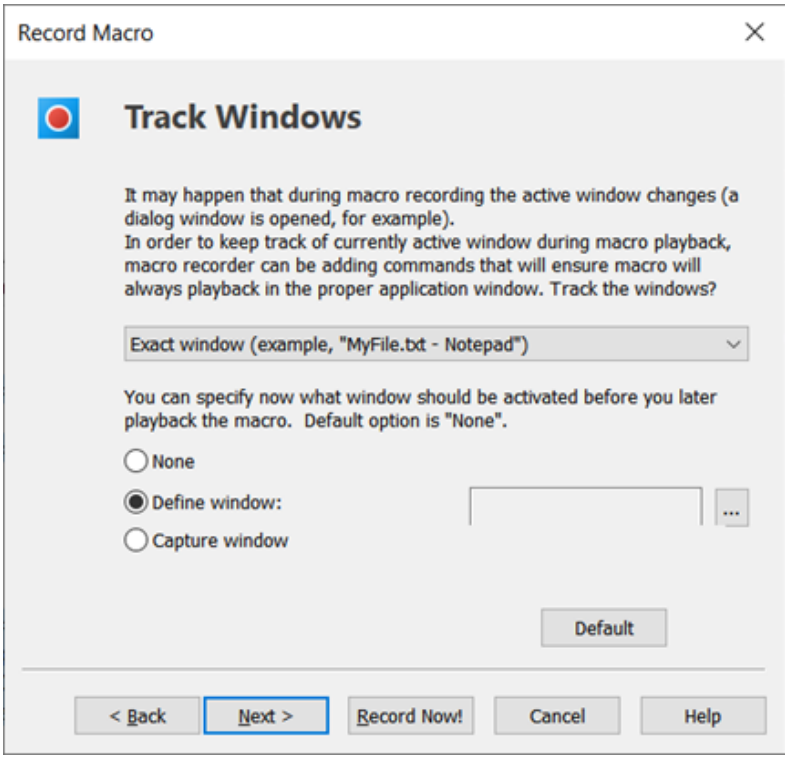


In this step, the user defines where to save the macro recorded in memory:

- **Add new item to the currently selected group** - new macro is added to the currently selected group.
- **Overwrite currently selected macro** - currently selected macro is overwritten by the new one recorded.
- **Add new macro to this group** - macro is added to the group selected using “Select” button.
- **Overwrite this macro** - macro selected using “Select” button is overwritten by the recorded one.

Use "Default" button to select default settings.

Track Windows



- **Track active window**
  For reliable macro playback it is necessary that the keystrokes and mouse clicks are played back in the correct window - the same window they were recorded in. If an active window is changed during macro recording (user switches from one program, for example Web browser, to other program, for example Excel) then this change can be captured in the macro and macro commands to activate proper window during macro playback will be added automatically to the macro. Windows tracking options are:
  - **No**
    Activated windows are not tracked at all. Recorded macro will blindly playback keystrokes and mouse clicks in currently active window.
  - **Generalized window**
    If the active window contains a file name (for example, "MyFile.txt - Notepad") then

the file name is replaced by * (for example, "* - Notepad"). This causes that the recorded macro will properly run in all Notepad windows regardless the file open - the macro will run the same in "MyFile.txt - Notepad" window as well as in the "Other File.txt - Notepad" window (however, it will not run "My document.doc - Word" window).

- **Exact window**
  The active window is captured exactly as it is (for example, "MyFile.txt - Notepad"). This causes that the recorded macro will properly run in "MyFile.txt - Notepad" window while it will NOT run in "Other File.txt - Notepad" window.

Define what window is automatically activated when macro playback starts:

- **None** - the macro playback will start in the currently active window. This option is used for macros that are intended to work in multiple different windows.
- **Define window** - defined window is activated before macro playback starts. Use [...] to define the window.
- **Capture window** - the window to activate is captured on the start of macro recording (the window where first keystroke or mouse click occurs).

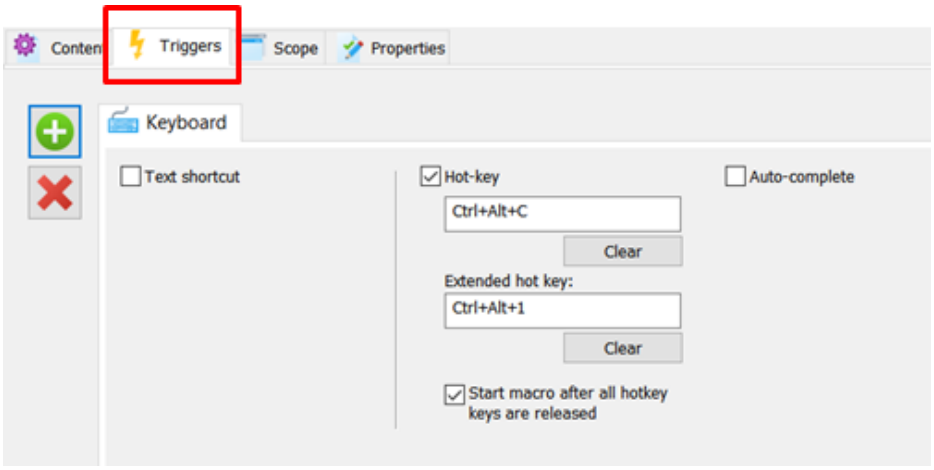
Use "Default" button to use default settings.

Ready to Start

Macro recording is started after user clicks “Record Now!” button. There is option to postpone macro recording using “Shift” key. If user holds “Shift” key down and clicks “Record Now!” button then the macro recording is started after “Shift” key is released. This allows user to use mouse to activate (bring to front) appropriate window for macro recording.

Triggers

Triggers define how to start macro execution. There are available these triggers:



- Keyboard Triggers Mouse Triggers
- Macro Toolbar Button
- Time/Date
- Window
- File
- Folder
- Drive
- Idle
- Windows Shutdown
- Display Pixel
- Clipboard
- Windows Service

Keyboard Triggers

There are three types of keyboard triggers:

- **Text shortcut**
  Text shortcut is a short text that starts macro execution when it is typed.

  - **Do not delete shortcut** - if checked the the typed text shortcut is not deleted. In other case the typed text shortcut is automatically deleted.

  - **Expand Automatically** - if checked then the macro execution is started immediately when the text shortcut is typed. In other case the macro is started after an expand key (space bar by default) is hit.
  - **Can be part of word** - if checked then the text shortcut can be part of a word (example: "MacroTool*works*" where "*works*" is text shortcut). If not checked then text shortcut starts macro only in case it is typed after a new line, space or other character or key that breaks continuous word typing (example: "MacroTool *works*").

  - **Case insensitive** - if checked then text shortcut starts macro regardless of capital non-capital letters.

  - **Follow case** - if case insensitive is checked and the macro text is just free text (or at least macro starts with free text) then the macro text is automatically modified this way:
    shortcut -> this is macro text
    Shortcut -> This is macro text
    SHORTCUT -> THIS IS MACRO TEXT

- **Hot-key**
  This is a hot-key trigger such as "Ctrl+Alt+T". When the defined key combination is pressed the macro execution is started.

  - **Extended hot-key** - This is a second hot-key that needs to be pressed in order to start macro execution. The purpose is to make it possible to use the same hot-key (for example, Ctrl+Alt+T) for multiple macros and distinguish among them by extended hot key. For example, "Ctrl+Alt+T" + "Ctrl+1" starts one macro, while "Ctrl+Alt+T" + "Ctrl+2" starts other macro, and so on.

  - **Start macro after all keys are released** - if checked then the macro execution is started after all keys (Ctrl, Alt, etc.) that are part of the hot-key are released (not pressed).

- **Auto-complete**
  This trigger only works for macros with just free text (or at least macro starts with free text). If checked, then in a case user starts typing text that is the same as (matching) the macro text then a hint window with macros list is shown. Once user pick the matching macro from the list the macro execution is started (and typically the macro text is written next to where user was typing).

Mouse Triggers

There are several triggers tied to different mouse events:

- **Mouse button click**
  Starts macro when defined mouse button (left, middle, right, x1, x2) is clicked. Required mouse location when button click occurs can be specified as:
  - Anywhere on screen
  - Anywhere in a window
  - On window title bar
  - On desktop
  - On Task-bar

  Keyboard control keys - Ctrl, Alt, Shift - can be required to be hold when mouse click occurs. In addition, other mouse buttons (middle, right, etc.) can be required to be hold when the left mouse button click occurs.

- **Mouse button double-click**
  Starts macro when defined mouse button is double-clicked. Other options are the same as for mouse click trigger.

- **Hold mouse button down**
  Starts macro when defined mouse button is hold pressed down for a second or two. Other options are the same as for mouse click trigger.

- **Shake horizontally**
  Starts macro when mouse is quickly moved in horizontal line from left to right and back several times. Required screen area can be specified as:
  - Anywhere
  - Upper screen area
  - Middle screen area
  - Lower screen area

  Keyboard control keys - Ctrl, Alt, Shift - can be required to be hold.
  In addition, mouse buttons (left, middle, right, etc.) can be required to be hold.

- **Shake vertically**
  Starts macro when mouse is quickly moved in vertical line from top to bottom and back several times. Required screen area can be specified as:
  - Anywhere
  - Left screen area
  - Middle screen area
  - Right screen area

  Keyboard control keys - Ctrl, Alt, Shift - can be required to be hold.
  In addition, mouse buttons (left, middle, right, etc.) can be required to be hold.

- **Mouse is moved to screen edge**
  Starts macro when mouse is moved to one of the screen edges:
  - Top
  - Bottom
  - Left
  - Right

Keyboard control keys - Ctrl, Alt, Shift - can be required to be hold.
In addition, mouse buttons (left, middle, right, etc.) can be required to be hold.

- **Mouse is moved to screen corner**
  Starts macro when mouse is moved to one of the screen corners:
  - Top left
  - Top right
  - Bottom left
  - Bottom right

  Keyboard control keys - Ctrl, Alt, Shift - can be required to be hold.
  In addition, mouse buttons (left, middle, right, etc.) can be required to be hold.

- **Mouse wheel**
  Starts macro when mouse wheel is rotated forward or backward. Required mouse location can be specified as:
  - Anywhere on screen
  - Anywhere in a window
  - On window title bar
  - On desktop
  - On Task-bar

  Keyboard control keys - Ctrl, Alt, Shift - can be required to be hold when mouse click occurs.
  In addition, other mouse buttons (middle, right, etc.) can be required to be hold when the left mouse button click occurs.

Toolbar Button

If a macro toolbar option is enabled on a macro group then all macros from this group are automatically showing up in the tool bar (and the macro group child groups as sub-menus). It is also possible to display a macro from one group in a macro tool bar that represents other group.

Button shows in tool bar - select a macro tool bar from the list or select <THIS> to display the button in the macro tool bar it belongs to.

Macro button - if selected, the macro is represented as a button in the macro tool bar and clicking on the button cause the macro is started.

Sub-menu button - if selected, a sub-menu button is showing in the macro tool bar. The sub-menu can be picked from the "Select" button.

See also:

- Run macro from Macro toolbar

Time/Date

This trigger type allows to schedule macro start to defined time and date.

Once - if this option is selected then the macro is run just once. After the macro finishes its execution it is automatically disabled to prevent it is started by this trigger again.

Every - if this option is selected then the macro is repetitively executed every second, minute, hour, day, week or month. "More options" button can be used to refine the configuration.

Start time - defines time and date when the macro should be started for the first time.

End time - defines time and date after which the macro should not be executed anymore.

Wake up computer if in stand by - if the computer is in sleep or hibernation mode when the macro should be started then the computer is automatically woken up, the macro is executed and the computer is put to sleep mode again.

Window

This trigger type allows to start macro when an application window is opened, activated, closed, etc.

- **Window state**
  - **Activated** - window is brought on top receiving keyboard input (window that user is currently working).
  - **Deactivated** - other window is activated so this one is not active anymore.
  - **Opened** - application is started and its window is created.
  - **Closed** - application is closed.

- **Window Identifier**
  A prescription that identifies the window. It consists from window title, window class and other characteristics. A [...] button can be used to edit the window identifier (typically "Select Window" button is used to extract the window identifier from an open window). Window identifier can contain * and ? wild-cards to allow more flexible matching of the window identifier to existing windows.

- **Exact match**
  If checked then the window identifier must match an existing window exactly. Otherwise just partial match (such as case insensitive) succeeds.

File

This trigger type allows to start macro when a file is created, changed, deleted, etc.

File event

- **Created** - file is created. The same effect has if an existing file is renamed.
- **Deleted** - file is deleted.
- **Changed** - file is changed (file attributes such as last modification time).
- **Bigger than** - file becomes bigger than predefined size.
- **Smaller than** - file becomes smaller than predefined size.

File - path to the file (such as: *c:\folder\myFileTowatch.dat*). If this is not full path (just *myFileTowatch.dat*) then path relative to the macro file location is considered. File path can contain Windows environment variables (for example: *%TEMP%\myFileTowatch.dat)* or Macro Toolworks application global variables (just simple, no array) - for example: *c:\folder\%ga_File%*.

Folder

This trigger type allows to start macro when a folder content changes.

Folder event

- **New file in the folder is created**
- **A file in the folder is deleted**
- **A file in the folder is changed**
- **New sub-folder is created**
- **A sub-folder is deleted**

Matching Folder Content

Folder path such as: *c:\folder*.txt* causes that if a text file in "*c:\folder*" folder is created, deleted or modified then the macro is started. If this is not full path (just **.txt*) then path relative to the macro file location is considered. File path can contain Windows environment variables (for example: *%TEMP%*.txt)* or Macro Toolworks application global variables (just simple, no array) - for example: *c:\%ga_Folder%*.txt*.

Save result files/folders to variable

This field contains the name of the variable (array) that receives list of files/sub-folders that matched the criteria and caused the macro to start. This variable can be used in the macro to manipulate the files/sub-folders (such as copy them, delete, etc.).

Save number of result files/folders to variable

This field contains the name of the variable that receives the number of files/sub-folders.

Drive

This trigger type starts macro when free size on specified drive is lower then limit.

Size in Gigabytes (GB)
Size in Gigabytes.

Drive
Drive to check. For example: *C:*

Idle

This trigger type starts macro when there is no user activity (keyboard, mouse) for certain time.

- **Minutes**
  Amount of time - in minutes - with no keyboard or mouse activity.

- **Computer**
  If this option is selected then the activity is for whole computer (all programs).

- **Window Identifier**
  A prescription that identifies the window to watch the idle activity for. It consists from window title, window class and other characteristics. A [...] button can be used to edit the window identifier (typically "Select Window" button is used to extract the window identifier from an open window). Window identifier can contain * and ? wild-cards to allow more flexible matching of the window identifier to existing windows.

Windows Shutdown

This trigger type starts macro when Windows is shutting down.

Display Pixel

This trigger type starts macro when a specified pixel on the computer screen changes color.

- **Pixel changes FROM color**
- **Pixel changes TO color**

- **Pixel RGB color**
  Color defined in RGB.

- **X,Y screen coordinates**
  Absolute screen coordinates.

  Use "Locate pixel" button to capture pixel color and coordinates.

Clipboard

This trigger type starts macro when clipboard content changes. One of these triggers can be selected:

- Clipboard data has changed or clipboard content was deleted
- Clipboard data has changed (but clipboard is not empty)
- Clipboard content is deleted and clipboard is empty

Windows Service

This trigger type starts macro based on the defined Windows Service state. The Windows Service name to test can be The condition

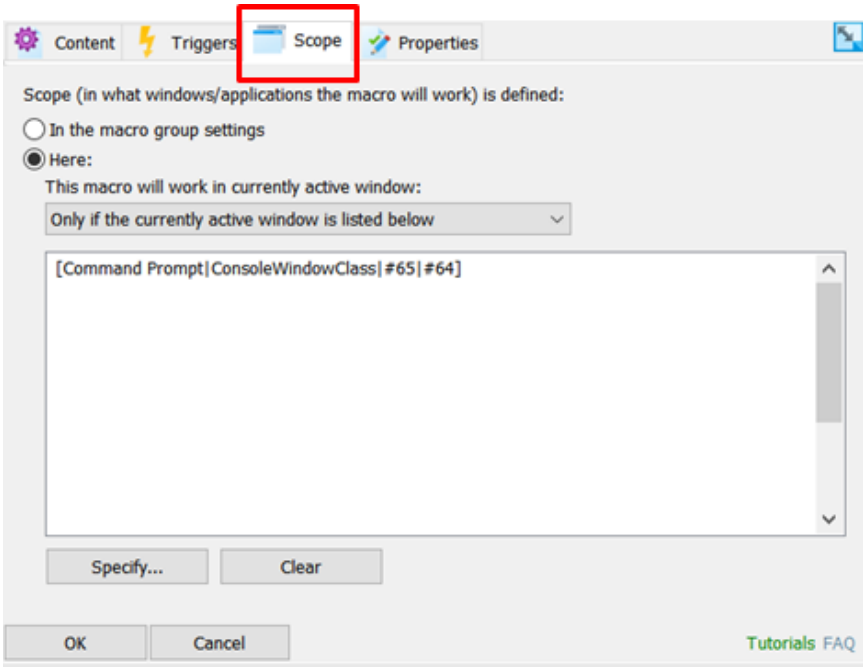
Run macro if Windows Service:
The name of the Windows Service to test for its state. The services names can be seen in Windows 10 "Task Manager" or in Windows "Computer Management", services section.

The states:

- **Running** - the macro is started if the service is running.
- **Not Running** - the macro is started if the service is installed but it is not running.
- **Installed** - the macro is started if the service is installed regardless it is running or not.
- **Not Installed** - the macro is started if the service is not installed.

Macro Scope

By default, macro triggers start macro in any application (window). However, it is also possible to make macro triggers to work only in defined windows. When such a defined window is active then user can use macro trigger to start the macro. If other window is active the trigger simply doesn't take any affect and macro is not started. This feature can be used to:



1. For different applications, create different macros that use the same trigger. (From the user's point of view, the macros, for example, will do the same thing but macro for each application will be programmed different way.)

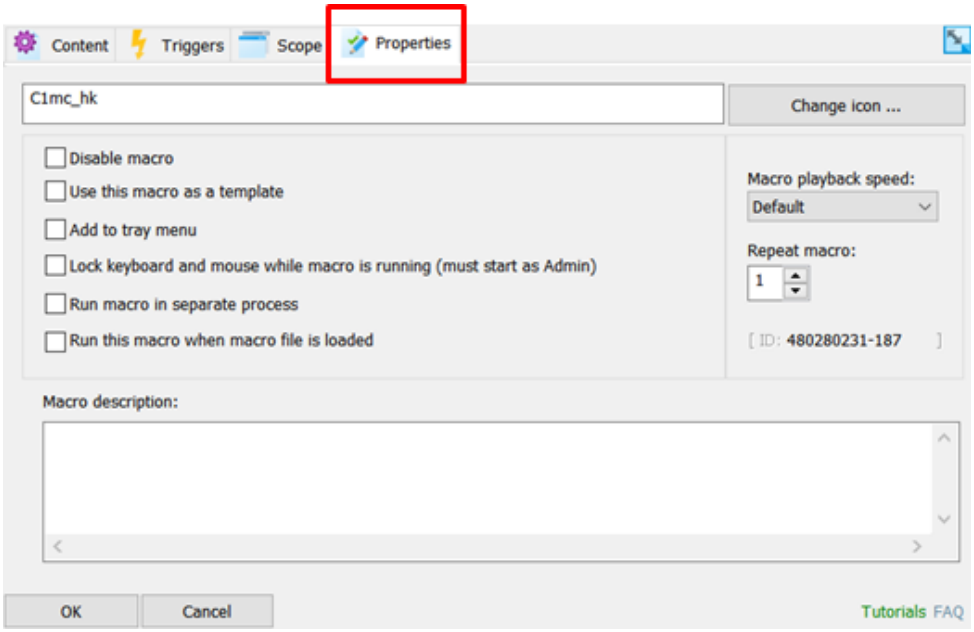
2. Create macro that works only in specific application(s) while the trigger doesn't take effect in other applications.

The scope options are:

- **In the macro group settings**
  If this option is selected then it is deducted from scope set in macro group if the trigger works in currently active window or not.
- **Here**
  Macro triggers scope is defined for this particular macro.
  - Always
  - Only if currently active window is listed below
  - Only if the currently active window is NOT listed below
- **Specify..**
  This opens up a window that allows user to configure window identification
- **Clear**
  Clears the window identification selected in the list

Macro Properties

Macro properties include these fields:



- **Macro name**
  The macro name is shown in the list of macros in the main window if the name column is enabled in program settings. The macro name is also used in:
  - Macro menus
    Macro menu shows up if multiple macros have the same trigger and the trigger is fired (such as multiple macros have Ctrl+Alt+T hot key assigned and the this hot key was hit). Macro menu also shows up when <macromenu> command is executed.
  - Macro toolbars as button text or tooltip

  For these two purposes above the macro name can:
  - Contain macro variables (application global variable or system variable - learn more here). For example, the macro name can be this: "Clipboard: %_vClpText%". The variable %_vClpText% in the macro name is in macro menu or toolbar expanded to text currently in clipboard.
  - Be empty. In such case in the macro menus or toolbars either macro description is displayed (if not empty either) or the macro text (script). This can be handy if the macro text (script) is just a simple text (text insertion macro).
- **Disable macro**
  If checked then macro triggers do not work and macro is removed from all macro tool bars (and menus).
- **Add to tray menu**
  If checked, the macro will be added to the tray menu (menu that appears when the program's tray icon is right-clicked) for quick macro start.
- **Use this macro as template**
  If checked, this macro will be available in "Add macro" drop down list. When a new macro is created from template then it means that a copy of template macro is simply created.

- **Run macro in separate process**
  See * below.

- **Run this macro when macro file is loaded**
  If checked, the macro is automatically executed when the macro file is loaded.

- **Lock keyboard and mouse while macro is running**
  If checked, the keyboard and mouse is locked during macro execution. This means that user cannot affect macro execution by moving mouse to other position or by hitting keys on the keyboard. From other hand, when this option is checked, it is not possible to stop macro execution by pressing "Shift+Esc" key combination. In a case macro is running and it is necessary to unlock mouse and keyboard again in order to stop macro execution then press Ctrl+Alt+Del, return back to desktop and use "Shift+Esc".

- **Macro playback speed**
  There are several options how fast the macro should be executed. This option makes only sense when the macro plain text is being sent as keystrokes and there is desire to slow up or speed down the keystrokes sending. Typically, this option can be useful for recorded macros.

- **Repeat macro**
  How many times the macro should be subsequently run.

- **ID**
  Unique identifier of the macro assigned automatically to the macro by program. The ID can be used to reference the macro.

- **Macro description**
  Free form text. Can be used for notes, description, TODO's etc.

---

* Run macro In Separate Process

During a macro execution no other macro can start. This is not convenient for macros that can take very long to finish (for example, FTP download). In such case there is an option to start time consuming macro in separate process so that it doesn't block other macros from starting.

Starting macro in separate process means, that a new process to run the macro is loaded to computer memory and the macro execution continues within this process. When macro execution is finished, the process is released from the computer memory again. There are three ways how the program can be started:

1. Normal (as currently logged user) - this is the default option.

2. Run as different user - this option allows user to start macro on other user's account. It is necessary to specify user name and password (and, optionally, domain). In addition, this option requires specific user rights to be set in Windows: Act as part of the operating system (SE_TCB_NAME), Bypass traverse checking (SE_CHANGE_NOTIFY_NAME) , Replace a process level token (SE_ASSIGNPRIMARYTOKEN_NAME), Increase quotas

(SE_INCREASE_QUOTA_NAME).
There is "WhoAmI.exe" utility that is part of the program distribution. You can create simple macro consisting just from <execappex>("whoami.exe","","",0,0) command. When the macro is started a WhoAmI.exe dialog box shows user name of the account under which the WhoAmI.exe program was started. This can be used to verify the macro is started on right account.

3. "As administrator" - this allows to run macro "as administrator" (with higher priority).

Run Macro

When macro is running, progress bar with “Stop” button is showing in lower right corner of computer screen (if this is not disabled program settings). User can click on "Stop" button and exit macro execution or click "Pause" button to pause and resume the macro execution. Macro execution can be also stopped by “Shift+Esc” key combination. The progress bar can be turned off in main settings dialog box.

There are many ways how to start macro execution:

- By Triggers
- From Main window, Tray, Run command, etc.
- From other program
- From macro toolbar

By trigger

See Triggers

From Main window, Tray, Run command, etc.

Except macro triggers, there are also other options how to run macros:

- **Tray Menu**

  User can right click on the program's tray icon and select from macros listed on the top of the menu shown. There are listed only macros with "Add to tray menu" option checked.

- **Main Window**

  The macro can be started from the main window using either toolbar button or "Tools/Run Macro" menu item.

- **All Macros List**

  The user can right click on the tray icon and select "All Macros List" menu item. User can select macro in the dialog box shown and click "Run" button.

- **From Link**

  User can create "Desktop Shortcut" (menu item "Tools / Create Desktop Shortcut") to an item and run the macro by double-clicking on the desktop shortcut.

- **Drag & Drop**

  You can drag an item in the main window and drop it on a window you want to run the macro in.

- **Run MACRO (<run>) Command**

  Macro can be run from other macro using Run MACRO (<run>)command.

From Other Program

There are several options how to run macro from other programs:

1. **Running macro saved in macro file:**
   It is possible to start macro from other program or using a command line (Windows "Run" dialog box or command prompt window). Just start Macro Toolworks with this parameters:

   -run:<NameOfTheMacroToRun> <MacroFilePath>
   *For example, "C:\Program Files\....*MacroToolworks.exe *-run:Update C:\Program Files\....\mymacrofile.4tw*
   This will cause the macro program is loaded to the memory, opens the file specified in the parameter and runs the defined macro. Then the program exits.

2. **Running a macro from currently opened macro file:**
   If the Macro Toolworks is running with a macro file opened, then it is possible to start a macro from an opened macro file using "**RunMacro.exe**" program. It takes macro name as a parameter and causes the macro is executed.

   Example: "RunMacro.exe mymacro" will cause the "mymacro" macro is started.

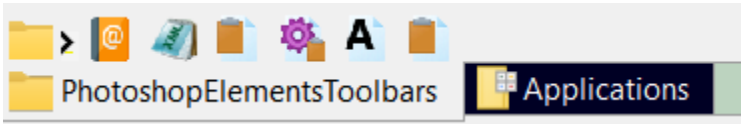
   Since both macro program and macro file are already loaded in computer memory the macro execution is significantly faster than case #1 above.

3. **Running macro saved in .MCR text file:**
   It is possible to save macro to a text file with "**.mcr**" extension (just right-click in macro steps editor and pick the command from menu). Macro that is saved in .MCR file can be started just by double-clicking in the Windows Explorer or other file management program. Running .MCR macro is faster then running the same macro using approach #1 above because it is not necessary to load a macro file (probably with many other macros defined). Note: Subsequent macro calls invoked by Run MACRO (<run>) command from the macro will fail. This means that an .MCR macro should not contain Run MACRO (<run>) command.

From Macro toolbar

It is possible to expose macros on the screen in user configurable tool bars for running macro by click on a button. There are two types of toolbars available:

- **Main** macro toolbar

  There is one tool bar that can have multiple tabs. Each tab represents one macro group. A presence of macros from a macro group in the tool bar is configured in macro group properties. The tool bar can be docked to a screen edge on any monitor (if there are multiple monitors connected to the computer). Auto-hide feature is supported on this tool bar. The tool bar is optional and can be turn on/off in program settings.

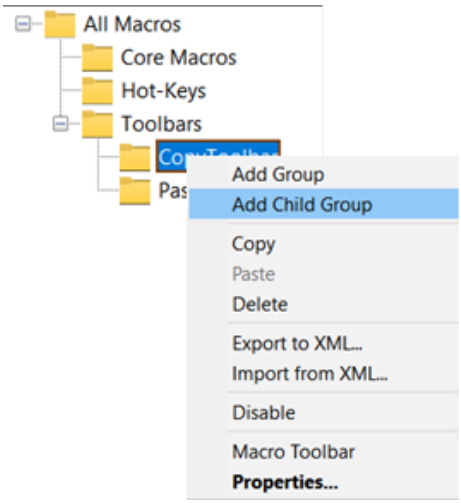
- **Flying** toolbar

  There can be multiple simple flying toolbars located anywhere on the screen. Each tool bar follows macro group scope defined in macro group properties - if currently active window fits to the scope defined for the macro group then the tool bar is showing on the scree; if the currently active window does not fit to the scope then the tool bar is not showing on the screen. A presence of macros from a macro group in the tool bar is configured in macro group properties.

Macro group

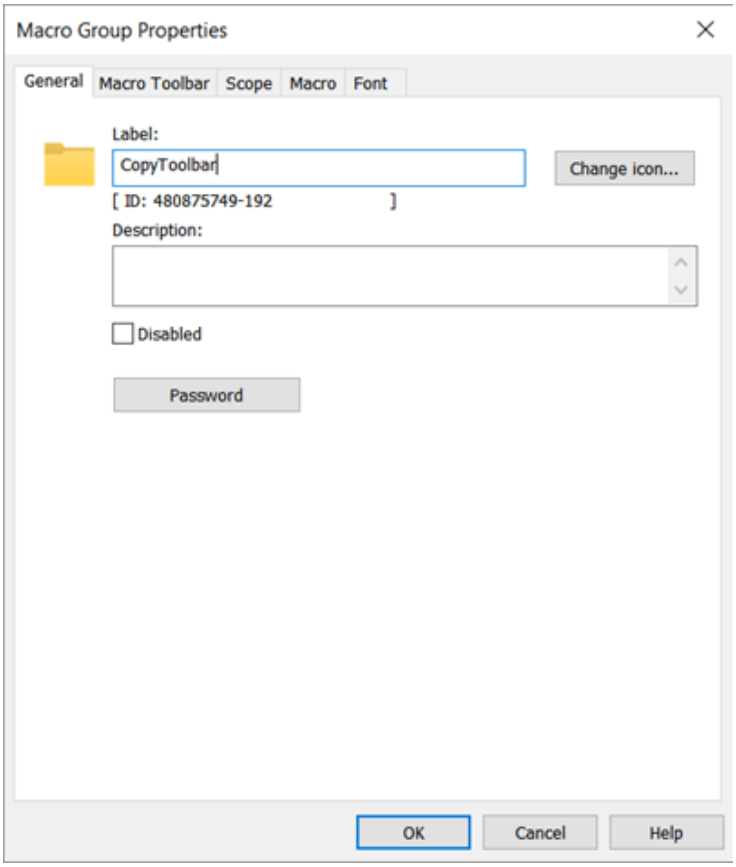
Macro Groups



Macro groups allow to organize macros in hierarchical structure. The hierarchical structure maintains top-to-bottom feature propagation. For example, if top-most macro group is disabled then all its child groups are behaving as disabled as well. The top-to-bottom feature propagation applies to properties like "Password", "Disabled", "Before/After", etc.

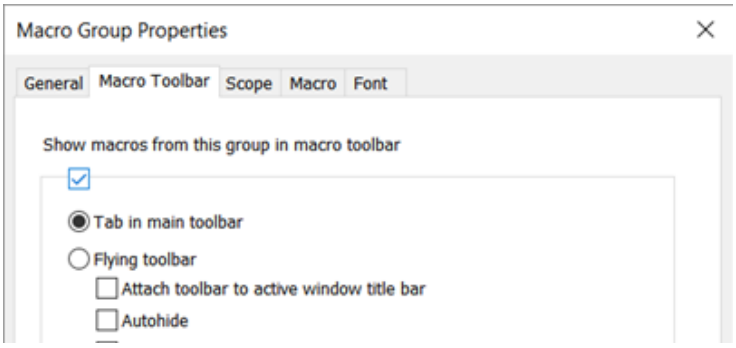
Macro Group Properties

General



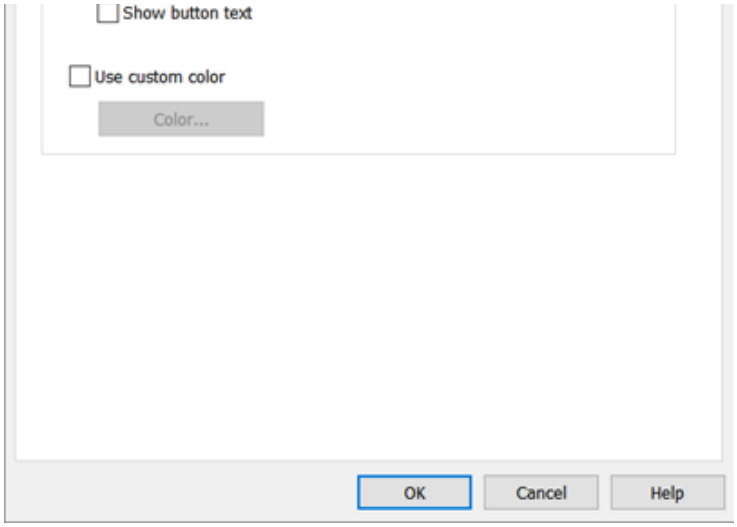
- **Label** - the name of the macro group
- **ID** - a unique identifier that can be used to reference macro group in some macro commands.
- **Description** - field for group description or notes.
- **Password** - button to password protect macros from this group (and child sub-groups).
- **Change icon** - button to select/change icon.
- **Disabled** - if checked, all macros from the given group (and child sub-groups) do not react to their defined triggers.

Macro Toolbar



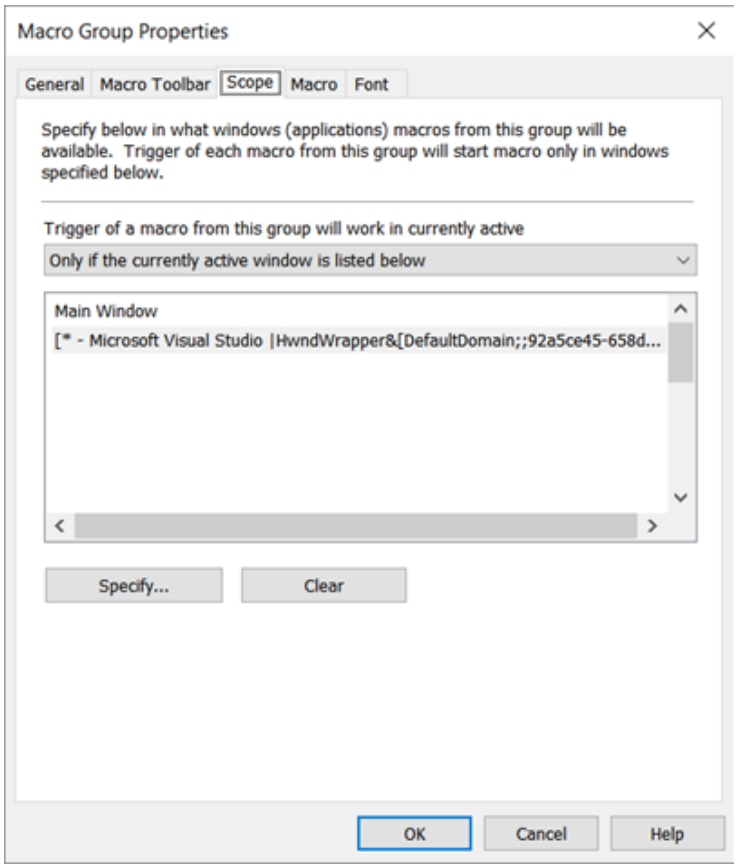
- **Enable** - if enabled then a macro toolbar representing this macro group is displayed on the screen and macros from this group are represented as buttons on the toolbar.
- **Tab in main toolbar** - if this option is

selected then the group is presented as a tab in main toolbar.



- **Flying toolbar** - if this option is selected then the group is presented as flying toolbar on the screen.
    - **Attach tool bar...** - if checked, the tool bar is showing around title bar of the window that is on top (receiving keyboard input).
    - **Auto-hide** - if checked, the tool bar is shrinking to minimal size when mouse is not on it.
    - **Show button text** - if checked, the macro name is showed next to button icon.
- **Use custom color** - if checked then the tool bar background color can be selected.

Scope Tab



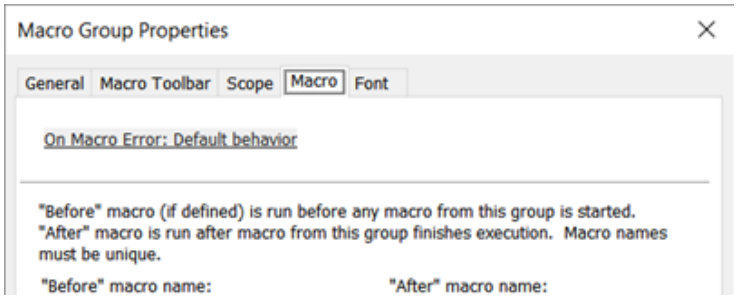
It is possible to make macros from the given group (and all child sub-groups) a window specific. This means that the macro triggers (including macro toolbars) will only start the macros in windows that belong to the defined scope. There are three options available:

- Macros are available in all windows (this is default).
- Macros are available only in windows user specifies.
- Macros are available in windows other then user specifies.

See also:

- Macro Scope

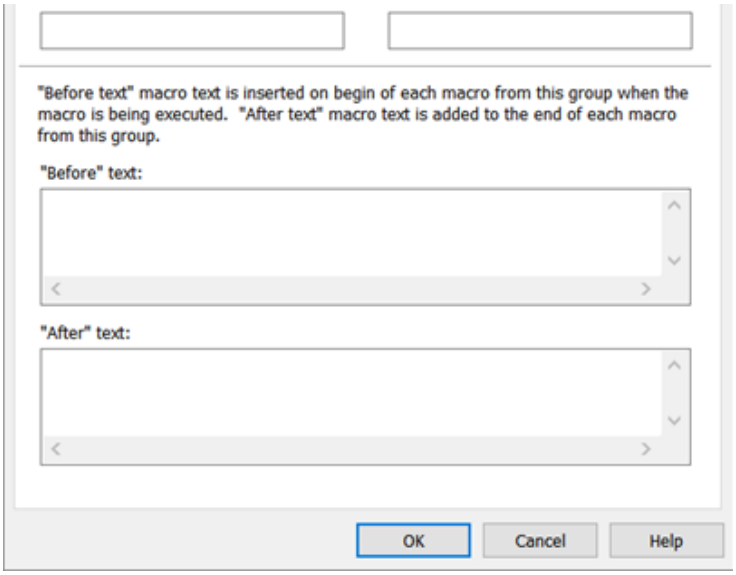
Macro



Before/After Macro

It is possible to configure so called "before" macro and "after" macro for each macro group. Names of existing macros go to

these fields (or fields can be left empty). The "before" macro is automatically executed before any macro from this group; the "after" macro is executed after execution of any macro from the group is finished. Example: Let's say we have a group with five macros named 1,2,3,4,5 and the macro 1 is defined as "before" macro and macro 2 is defined as "after" macro. Now when user starts macro 4, for example, the macro 1 is executed, then macro 4 is executed and then macro 2 is executed.



Before/After Text

It is also possible to use "before text" and "after text" fields to achieve the same result. The difference is that macro text (macro steps) are put to these fields instead of names of existing macros.
The "before" and "after"feature simplifies macros creation in cases when multiple macros do the same steps at the beginning/end of the macro execution (such as some initialization, conditions checking, clean up steps, or including (<-include-> command) a macro file with predefined procedures).

See also:

- **On Macro Error**

Font

User can select other font to be used in general macro code editor and in the macro list in the main window.

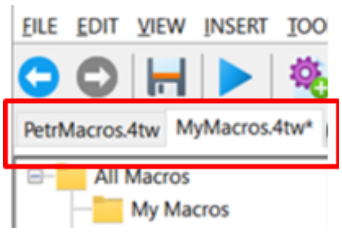
Macro File

Macros are stored in a single macro file.

- Macro File Tab
- Create/Open/Save
- Import / Export
- Read-only / Read-write

Macro File Tab

Each macro file that is open in Macro Toolworks is represented by a tab. By switching the tab it is possible to edit particular macro file content. It is possible to right-click the tab in order to close it or to navigate to the folder where the macro file is located.



Create/Open/Save

The macro files are by default located in "Documents\MacroToolworksFiles\MacroFiles" folder. A new file can be created using “File/New” menu command, an existing file can be open by clicking on “File/Open” menu command. All macros are saved within a single file which makes distribution of macros to other computer an easy task. Just copy (or use "File/Save as" menu command) the macro file to a removable media and then copy it back to hard drive on other computer and use "File/Open" menu command to open the macro file.

It is possible to have open multiple macro files in the program. Each macro file is represented by a tab in the window. Clicking on the tab displays the content of the given macro file - macro groups and macros.

Import / Export

It is possible to export macro file in XML format and import it in other macro file. The export/import feature is available on:

- **Macro file**
  Use "File/Export to XML" menu command to export whole the file content.
- **Macro group**
  Right-click on a macro group and select "Export to XML" to export just this macro group.
- **Macro**
  Right-click on a macro and select "Export" menu item to export (all) selected macros.

Read-only / Read-write

If macro file has read-only access set then the content of the macro file cannot be modified. This feature is used when there is a need to share macro file among multiple users and prevent it from unauthorized modifications. Typically, macro files that are shared among many corporate users are located on shared drives with properly (read-only) set access rights.

File Backups

Previous Version File Backup

If a new version of the Macro Toolworks is installed then the macro files created in previous version are backed up when they are open in new version. The backup file is located in the same folder as the original file and its name has "_backup_of_verXXX" suffix (for example, "MyMacros_backup_of_ver862.4tw).

Previous Save File Backup

When macro file is saved then the current macro file is backed up before the new content is actually saved. So there are two latest versions of the macro file always available. The previous version of the macro file has the same name and .4tw.pre_XXX extension (for example, MyMacros.4tw.pre_862). Both versions of the file are located in the same folder.

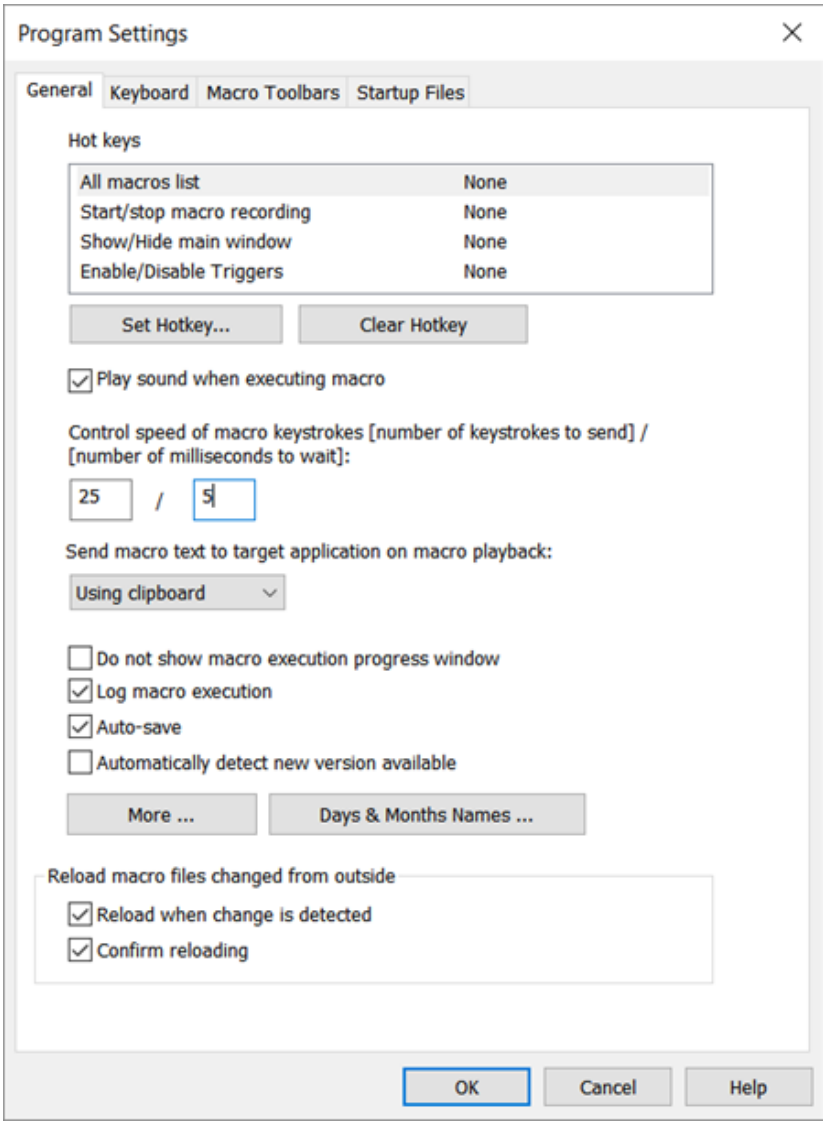
Program Settings

There are several program settings user can configure. The settings apply to all open macro files.

- General Settings
- Keyboard Settings
- Macro Toolbar Settings
- Startup Files

General

General



- **Hot keys:**

  Select the hot-key you want to set in the list and click "Set Hotkey..." button

  - **All macros list** - hot key that brings up window with list of all macros.
  - **Enable/Disable Triggers** - hot key to enable/disable macro triggers.
  - **Start / stop macro recording** - hot key to start/stop macro recording.
  - **Show / hide main window** - hot key to show or hide main window.

- **Play sound when executing macro** - if checked, defined sound (.wav file) is played when macro is started. A dialog box to define sound file is shown when the unchecked option is changed to checked. If no sound file is defined then a default sound is played.
- **Control speed of macro keystrokes sending**
  This option allows to control speed of keystrokes sending to target application. User may need to modify this option if, for example, macros are mostly run in target applications running on a remote machine accessed using remote desktop connection (or Citrix). In such case if keystrokes are send too fast then some of them can get lost. This option allows to define the number of keystrokes and a delay in milliseconds to wait with sending other keystrokes. For example, if the number of keystrokes is 25 and the delay is 20 then if there is a macro that is sending bigger amount of keystrokes then after each 25 keystrokes executed there will be 20 milliseconds delay.
- **Send macro text to target application...**
  This option allows to choose how simple macro text is inserted into the target application. If "As keystrokes" is selected then the simple macro text is sent to the target application (such as word editor) as a set of keystrokes the same as if the text is typed on keyboard.

If "Using clipboard" option is selected then the macro text is copied to the clipboard first and then it is pasted to the target application the same as if the user hits Ctrl+V key combination. The Macro Toolworks has no control on the target application. It can happen that the paste operation takes in the target application longer then expected and a subsequent macro that is eventually run precede the paste operation completion and cause unexpected result. To prevent this it is possible to set a delay in milliseconds that prevents other macros or commands to execute. This slows down the macro execution but increases the reliability.

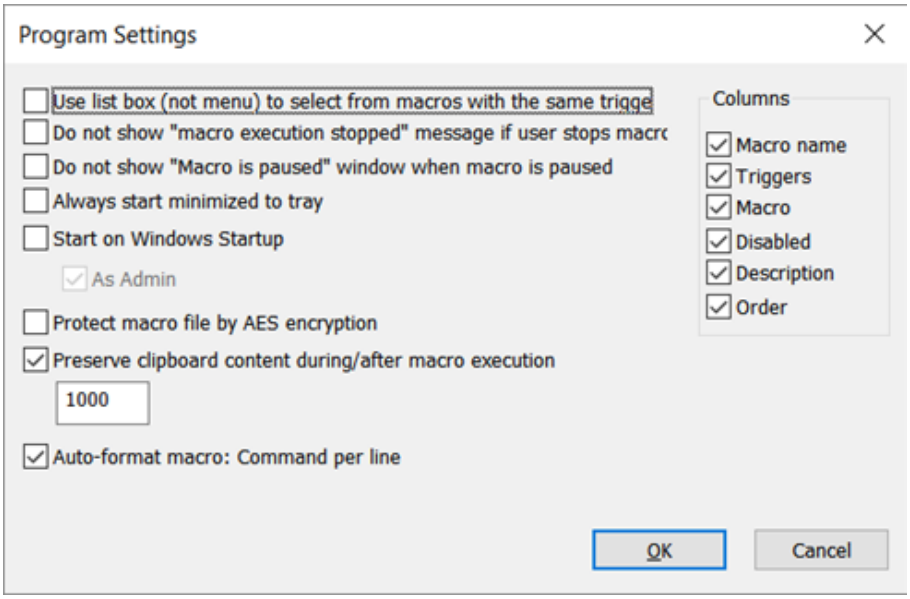
- **Do not show macro execution progress window** - if checked, the macro execution progress window is not showing.
- **Log macro execution** - if enabled, macro execution information is added to the log.txt file. The file is available from the "View/Show Log File" main menu item. The log file collects macro execution start time, finish time, and errors that occur during the macro execution.
- **Auto-save** - if checked, the changes made to macro file are automatically saved every couple minutes. In addition, all changes are automatically saved on program shutdown (without showing "Do you want to save..." dialog box to user).
- **Automatically detect new version available** - if this option is enabled, the program connects to its home page and detects a new version availability.

Reload macro files:

- **Reload when change is detected** - if a change in a macro file is detected, the macro file is automatically reloaded so that users always have the most recent version of the macro file laoded.
- **Confirm reloading** - if checked, the user is asked to confirm macro file reload.

More options button:

More options are available on [More...] button:

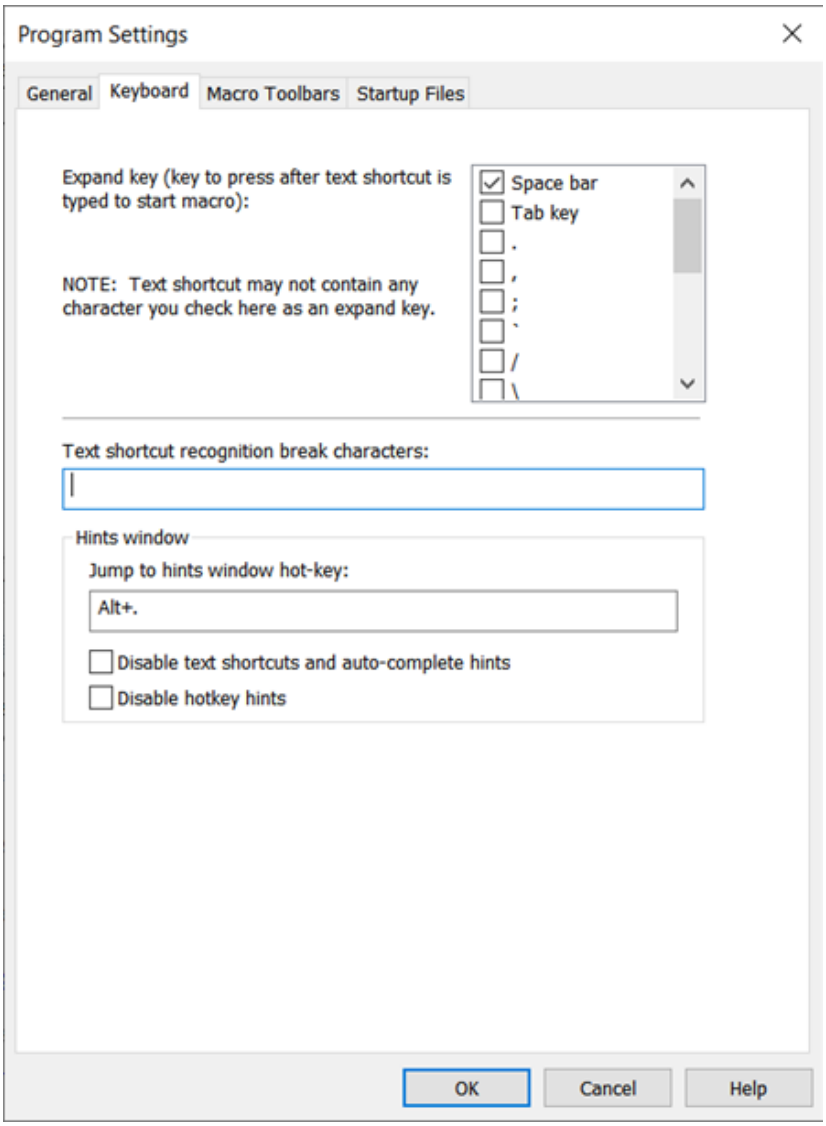


- **Use list box to select from macros with the same trigger** - It is possible to define multiple macros with the same trigger. This option tells if list box or menu should be used to select what macro will run.
- **Do not show "Macro execution stopped" message...** - It is possible to stop macro execution using "Shift+Esc" hot-key. By default, if the macro is stopped this way, a message is shown to user. This option allows to disable the message.
- **Do not show "Macro is paused" window...** - if checked, a message that macro is paused is not showing.

- **Always start minimized to tray** - if checked, the program always starts minimized to the Task bar tray icon. Otherwise the program starts in the same position as it was before exiting.
- **Start on Windows Startup** - if checked the programs automatically loads on Windows startup.
    - **As Admin** - if checked the program starts with admin privileges. This is a recommended option since some software (such as browsers) often blocks the keyboard triggers (such as text shortcuts) if the Macro Toolworks is not running with the admin privileges.
- **Protect macro files by AES encryption** - if checked, then all macro files content is encoded using AES algorithm for better security of the data saved in the macro file.
- **Preserve clipboard content...**
  If "Using clipboard" option is selected in "Send macro text to target application..." above then clipboard is used to send simple macro text to target application which causes the clipboard content is modified. This option allows to preserve the clipboard content so that after a macro execution is finished the clipboard content is restored to be the same as it was before the macro execution was started. It is possible to set the delay in milliseconds how long the Macro Toolworks is waiting before the clipboard content is restored. If the delay is too short then it can happen that before the target application is able to react to Ctrl+V paste operation the clipboard is restored and the original clipboard content is inserted to target application instead of the macro text. If the delay is too long then if user wants to paste clipboard data immediately after executing a macro the macro text is again pasted instead of the original clipboard content. A value in range 1000 (1 second delay) to 3000 (3 seconds delay) should work fine in most cases.
- **Auto-format macro: Command per line** - if checked, the macro - when edited as "Macro Steps" - formats the macro text so that there is one macro command per line. The formatting is applied when switching from "Macro Steps" tab to "Macro Text" tab.

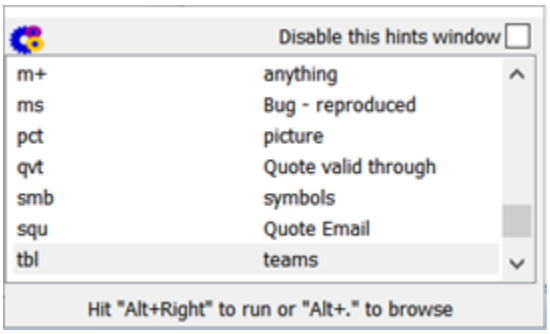
Keyboard

Keyboard



- **Expand key** - check the keys you want to use to start text shortcut expansion. (The keys that need to be hit after a text shortcut is typed)
- **Do not delete expand key** - if checked the expand key is not deleted.
- **Text shortcut recognition break characters** - the characters listed here cannot be part of text shortcut. This means that text shortcut recognition is restarted when such character is typed. Some keys causes this automatically (for example, page up, page down, tab, arrow keys, enter, etc.) but additional characters can be specified here. For example, we have "sct" text shortcut. When typing "(sct " the "sct" shortcut is correctly expanded only if the "(" character is listed in "non-text shortcut characters" field.
- **Hints Window**
  - **Jump to hints window hot-key**
    Hot-key that allows to jump to showing hint window to navigate among items listed. "Alt+." is the default hot-key.
  - **Disable text shortcuts and auto-complete hints**
    There is a small window that appears any time when the user is typing a text that (partially) matches defined text shortcut or auto-text. The window contains all the auto-text items and text shortcuts that are in the scope of the window (application) the user is typing in. The best fitting item is being preselected in the list and user can start it by pressing "Alt+Right Arrow" key combination. The user can also jump into the hints window using a user defined hot key (see above) and navigate in the list using "Up" and "Down" keys.

    [ ] If this disable option is checked then the hints window is not showing.

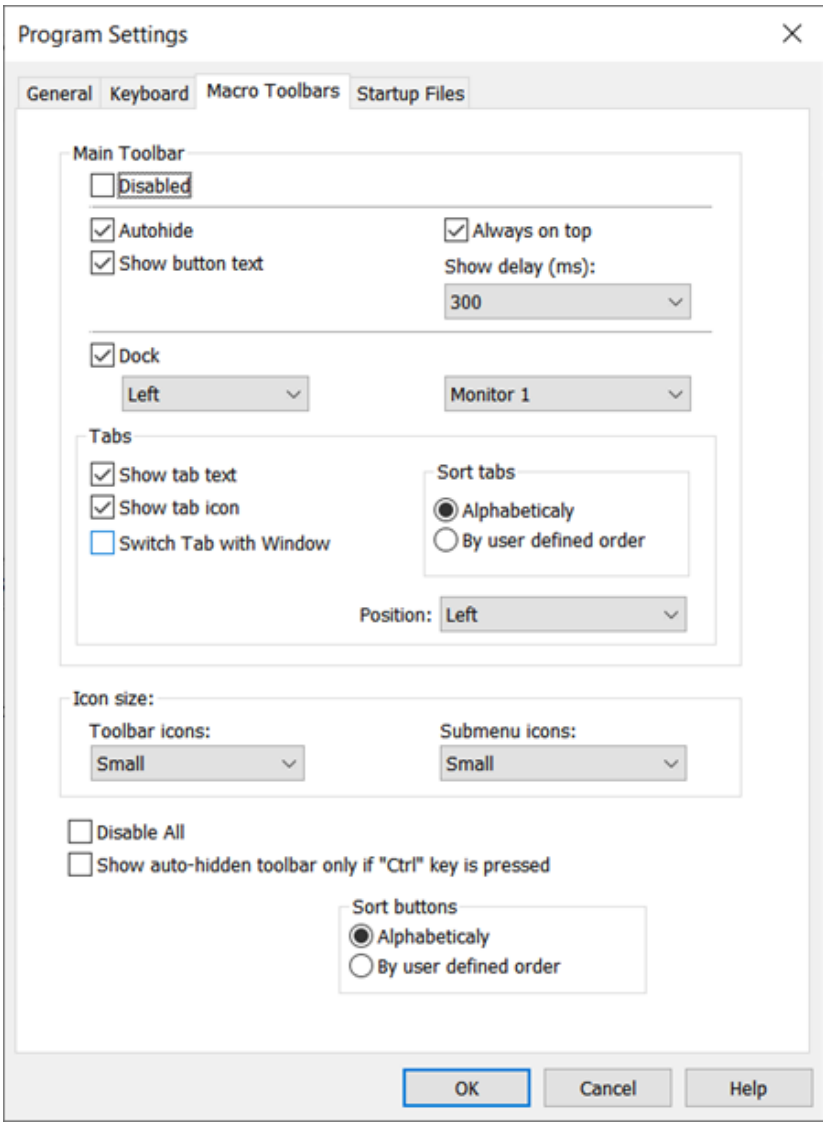


- **Disable hot-key hints**
  This is the same small window as described above. It appears any time user presses a control key (for example "Ctrl" key) and such key is part of a macro hot-key available (in scope) in the currently active application. Again, all available hot-keys are listed and best fitting one is preselected.

  [ ] If this disable option is checked then the hints window is disabled and is not showing.

Macro Toolbar

Macro Toolbars



- **Main Toolbar**
  - **Auto hide** - if enabled, the toolbar is hidden if the mouse cursor is out of the toolbar area. If the toolbar is not docked it is hidden to the toolbar edge that is closest to a screen edge. It means that it can be hidden to any of the top, right, bottom or left edge depending on the toolbar position on the screen.
  - **Disabled** - when checked, the main toolbar is not showing.
  - **Always on top** - if enabled, the toolbar is top most window that is never overlapped by other windows.
  - **Show delay** - allows to configure show delay time for auto-hide.
  - **Show button text** - if checked, macro name is showing along with macro icon (button icon).
  - **Dock** - if checked, the toolbar is "docked" on a selected screen's edge.
  - **Monitor** - allows to select on which monitor the toolbar is showing.
  - **Tabs**:
    - **Icon sizes** - allows selecting between small and large icons both for toolbar and submenus.
    - **Show tab text** - if checked, the tab name is displayed within each tab.
    - **Show tab icon** - if checked, the macro group icon is displayed on tab.
    - **Switch Tab with Window** - if checked, then tab is made automatically active along with a change of active window in case that macro group that the tab represents has the newly active window defined in scope.
    - **Tabs position** - tabs can be positioned on top or bottom for horizontal toolbar or on left or right for vertical toolbar.

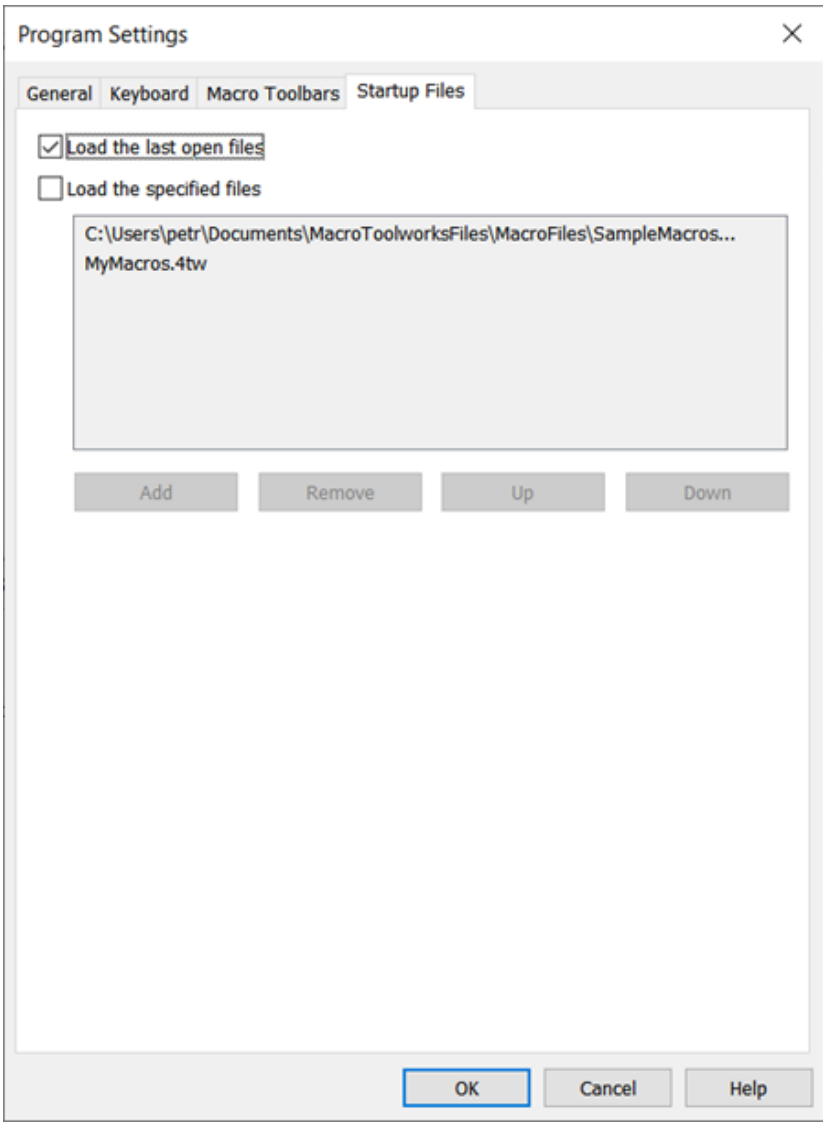
- **Sort tabs** - defines if the tabs are sorted alphabetically or by the user-defined order (order of the macro groups representing the toolbars).

All toolbars

- **Icon size** - allows to select size of the icons in the toolbars.
- **Submenu icons** - allows to select size of icons in the submenus.
- **Show auto-hidden toolbar only if "Ctrl" keys is pressed** - if checked, then to show auto-hidden toolbar it is necessary to move mouse to the little toolbar residua and also have "Ctrl" key down.
- **Sort buttons** - defines if the buttons are sorted alphabetically or by the user-defined order (order of the macros in the macro list defined by the "order" column).

Startup Files

Startup Files



These options allow to define what macro files are automatically loaded when the Macro Toolworks starts.

- **Load the last open files**
  If this option is checked then the files that were open before the program was shutdown or Windows were shutdown are open.

- **Load the specified files**
  If this option is checked then it is possible to specify the files that will be automatically open when the program is started. "Add" button allows to add a new macro file. If it is needed to specify a path relative to the default macro file location ("C:\Users\<USERNAME>\Documents\MacroToolworksFiles\MacroFiles" - see more here) then after a macro file is added using "Add" button hold mouse down on the newly added record (or use F2 key) and then edit the record appropriately (for example, change the record to "CorporateMacros\letterTemplates.4tw" in order to have open macro file "C:\Users\<USERNAME>\Documents\MacroToolworksFiles\MacroFiles\CorporateMacros\letterTemplates.4tw".
  The macro files are open in the same order as defined in the list. Use "Up", "Down" buttons to change order of the macro files if needed.

Security

There are several areas related to user data security:

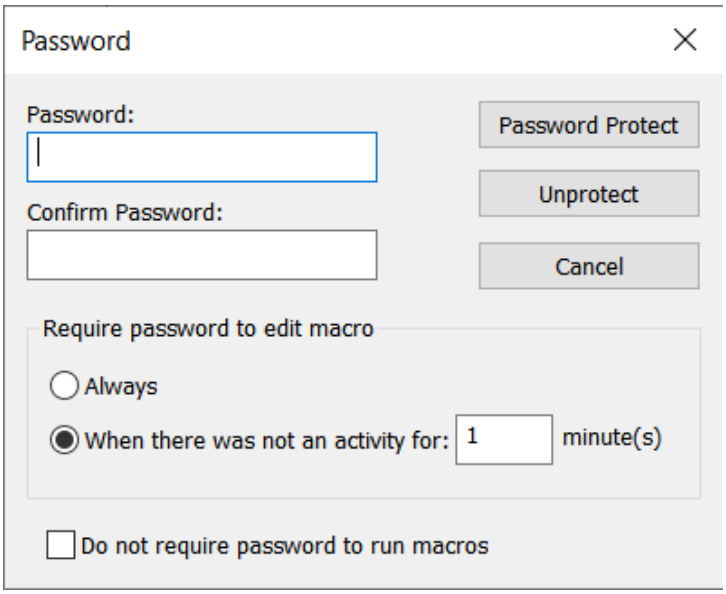
- Lock for Editing
- Password Protection
- File Data Security

Lock for Editing

Each macro file can be locked for editing so that changes in the file can be made only if the correct password is provided. This can be used to prevent macro file from changes when macro file is shared among multiple users. To lock the macro file use "File/Lock for Editing" menu command in the main window.

Password Protection

It is possible to password protect macros so that no one who doesn't know the password can see the macro definition, edit it or run it. The password protection is implemented on the group level. This means that user defines password for the given group and all the macros from this group are password protected. To define password, go to the macro group properties window and click "Password" button.



- **Password**
  Insert here the password to use.

- **Confirm Password**
  Type the password to use again here.

- **Always**
  If selected, the password is always required to edit macro.

- **When there was not...**
  If selected, the password is only required to use or edit macro if user did not run or edit a macro for specified amount of minutes.

- **Do not require password...**
  If checked, the password is not required to run macros (the password is only required to edit macros).

- **Password Protect**
  When this button is clicked then the dialog box is closed and the configured options are applied.

- **Unprotect**
  When this button is clicked then the password is not required anymore.

File Data Security

The file is encrypted using in-house encryption technology so that its content cannot be viewed in any file viewer or other program. In addition, the "Professional" edition of the program also allows to encrypt macro file data using AES industry standard cryptography algorithm. The AES encryption is indicated by "AES:Yes" in lower-right area of the main window.

Installation

The program installation related information:

- Default installation folder
- Silent Install
- Install on Shared Drive

Default Installation Folders

Program Installation Folder

Since the program version 8.2.0, the program is by default installed to the 32-bit Program Files folder (typically "C:\Program Files" on 32-bit Windows and "C:\Program Files (x86)" on 64-bit Windows).

User Data Folder

Since the program version 8.2.0, the user program settings and macro files are stored in "MacroToolworksFiles" folder in user documents folder (typically "C:\Users\<USERNAME>\Documents\MacroToolworksFiles"). This folder is created automatically when the program is executed for the first time after installation. The macro files (for example the demo file) are by default located in "MacroFiles" sub-folder, however, they can be saved to any other location using "File/Save as" menu command.

Silent Install

The installation program accepts optional command line parameters that can be useful to system administrators to silently install the program.

/SILENT

Instructs installation program to be silent. When installation is silent then the wizard is not displayed but the installation progress window is showing on the screen. Error messages (if any) are displayed as well as startup prompt (if it is not disabled by the '/SP-' explained above).

/VERYSILENT

The same as '/SILENT' above, just without installation progress window showing.

/SUPPRESSMSGBOXES

Instructs installation to suppress message boxes. It has effect when combined with '/SILENT' or '/VERYSILENT'.

The default response in situations where there's a choice is:

- Yes in a 'Keep newer file?' situation.
- No in a 'File exists, confirm overwrite.' situation.
- Abort in Abort/Retry situations.
- Cancel in Retry/Cancel situations.
- Yes (=continue) in a DiskSpaceWarning/DirExists/DirDoesntExist/NoUninstallWarning/ExitSetupMessage/ConfirmUninstall situation.

/LOG

Instructs installation to create a log file in the user's TEMP directory detailing file installation and [Run] actions taken during the installation process. The log file is created with a unique name based on the current date.

/LOG="filename"

The same as /LOG above, except it allows to specify a fixed path/filename to use for the log file. If a file with the specified name already exists it will be overwritten. If the file cannot be created, Setup will abort with an error message.

/NOCANCEL

Prevents the user from aborting installation process.

/CLOSEAPPLICATIONS

Instructs installation to close the program if it is running.

/RESTARTAPPLICATIONS

Instructs the installation to restart the program if possible.

/NORESTARTAPPLICATIONS

Prevents Setup from restarting applications. If /RESTARTAPPLICATIONS was also used, this command line parameter is ignored.

/DIR="c:\dir"

Overrides the default installation directory. A fully qualified pathname must be specified.

/GROUP="Group name"

Overrides the default group name.

/NOICONS

Instructs installation not to create desktop icon.

Install on Shared Drive

The program can keep multiple users settings while it is installed on a shared drive and all users starts the program from the shared drive. This feature allows multiple users to share one installation and minimize maintenance effort for corporate administrators (makes upgrading easy even if there are many users on the net).
Information about all the users is stored in the **users.ini** file that resides in the same directory where the product is installed. The file is a text file that can be edited by any raw text editor like Notepad. The file structure is as follows:

[header]
num_of_users=2

[1]
name=oscar
path=\\server\users\oscar

[2]
name=debie
path=\\server\users\debie

Note: The path name may not end with \ character! The [1], [2], etc., sections have to start by [1] and end by [num_of_users]. The number of users is limited by your license.

Starting Program

There are two ways how to start the program on user work station with user's settings loaded:

1. PATH_TO_THE PROGRAM /user:?
   Where the PATH_TO_THE PROGRAM is full path to the software executable. When the program starts with this "/user:?" parameter a login dialog expecting user name (from the users.ini file above) to be inserted appears.

2. PATH_TO_THE PROGRAM /user:USER_NAME
   Where the PATH_TO_THE PROGRAM is full path to the software executable and USER_NAME is the user name from the users.ini file above. This will start the program without any login dialog.

Adding New User

To add new user follow these steps:

1. Create new user's directory where the program can save settings and data files. You

can create a copy of the "*UserDir-Template*".

2. Increase the number of users in *[header]* section in users.ini file.

3. Add new user section (for example, [10] if there are ten users) and specify users name and the user's directory path (as per 1 above):

```
[10]
name=hugo
path=\\server\users\hugo
```

4. On the user's workstation, create a link to the executable command line as described in "Starting Program" section above.

Drag & Drop

Drag & Drop operation is supported across the program. It can be used to:

- Change macro groups structure
- Change order of macros in the macros list
- Move/copy macro from the list to other macro group
- Move macro commands in macro steps editor

Log file

The program is logging activities (such as what macros were running, macro failures, etc.) to a text log file. The log file can be open using "Show/Log file" menu command. It is possible to enable/disable logging in "Program Settings".

HTML Export/Print Macros

There is not a print command itself build in the product. Instead, it is possible to export whole the macro file content to an HTML using "**File/Export to HTML**" menu command. The exported file then can be viewed in a web browser and printed from there.

Export to HTML:

- **HTML File -** Generated HTML file name (path).
- **Include macro content -** if checked, also macro steps (macro text) is included to the HTML output.
- **Selected group only -** if checked, only macros from currently selected group are exported to the HTML file. Otherwise whole the macro file content is exported.

Generate Free Macro Player / EXE File

In the Professional edition of the software, it is possible to generate Free Macro Player macro file or an executable file (.exe) that can be distributed to other users so that they can use macros without a need to have Macro Toolworks installed. The file is generated from the macro file that is currently active (the selected Macro File Tab). There are these options:

1. Generate Free Macro Player file (.fmp)

Users need to have Free Macro Player software installed to be able to use the generated .fmp. The Free Macro Player is free of charge for all users (including business use) and it can be freely redistributed.

2. Generate Executable File (.exe)

An executable (.exe) file is generated. When the executable is run then it behaves so that Free Macro Player is started with the macro file loaded. It is the same as if user installs Free Macro Player and opens the file generated from options #1 above.

- *Password*
  It is possible to protect the generated file using password.
- *Program Settings*
  Program settings can be set for the generated .exe.
- *Package all files*
  If checked then whole folder (including sub folders) where the macro file is located is packaged within the generated .exe file. This allows to easily distribute additional data needed in macros.

3. Generate Executable File (.exe) that runs a single macro

This options allows to generate an executable (.exe) file that runs just one selected macro.

- *Password*
  It is possible to protect the generated file using password.
- *Program Settings*
  Program settings can be set for the generated .exe.
- *Macro to run*
  Insert the name of the macro that will be run when generated .exe file is executed.
- *Package all files*
  If checked then whole folder (including sub folders) where the macro file is located is packaged within the generated .exe file. This allows to easily distribute additional data needed in macros.

Build-in Hotkeys

The program has built-in hot-keys:

Hotkey	Action
Ctrl+N	Create new macro file.
Ctrl+O	Open a macro file.
Ctrl+S	Save changes made to currently opened macro file.
Ctrl+C	Copy.
Ctrl+V	Paste.
Del	Delete selected macro(s)/group.
Ctrl+F	Show list of all defined macros.
F3	Find text in the macros.
Ctrl+I	Add macro.
Ctrl+L	Add clipboard macro.
Ctrl+T	Create new macro from template.
Ctrl+R	Record new macro.
Ctrl+D	Add group.
Ctrl+G	Run selected macro.
Ctrl+A	Select all macros (in the macro list pane).
Ctrl+P	Open main settings dialog box.
Alt+E	Edit the current command (in macro text editor pane).
Alt+A	Add currently selected command (in commands and system variables tre
F11	Maximize/Restore macro steps editor.
Alt+D	Swap to macro steps editing.
Alt+R	Swap to macro text editing.
Alt+O	OK & save changes made in macro editor.
Alt+C	Cancel & disregard changes made in macro editor.
Alt+right arrow	Jump to next command (in macro text editor).
Alt+left arrow	Jump to previous command (in macro text editor).
Alt+up arrow	Jump to previous macro.
Alt+down arrow	Jump to next macro.
Ctrl+Page Down	Switch between tabs in macro editor.
Ctrl+Page Up	Switch between tabs in macro editor.
F5	Start macro debugging.
F10	Execute next command (in debugging mode).
F9	Show variable content view window (in debugging mode).

Icons Overlay Images

There are little icons that indicate some specific state of a macro or macro group:

Icon	Meaning
	Macro or macro group has defined scope - macro triggers run only defined set of applicat
	Macro or macro group is disabled.
	Macro group is password protected.

API's for External Programs/Scripts Interaction

There are several options how other programs and scripts can interact with the Macro Toolworks Professional edition (or Free Macro Player).

- HTTP API
- Command Line Executable
- Windows Script (WScript)

Http API

The Macro Toolworks Professional edition (and Free Macro Player) program supports (starting version 9.2.0) an HTTP API. It allows other programs to set or retrieve a value of an application global variable (the one that starts with "ga_" prefix - see more here - or a system variable - see more here), and start an existing macro and retrieve its result.

HTTP Ports

The program starts an HTTP server that listens (for localhost / 127.0.0.1 only) for HTTP requests on the following ports:

Program	Ports
Macro Toolworks	29592 - primary port 40804 - secondary port if the primary port is in use by other program 45054 - other port if the primary and secondary port is in use by other program
Free Macro Player	29593 - primary port 40805 - secondary port if the primary port is in use by other program 45055 - other port if the primary and secondary port is in use by other program

The actual port (most likely primary port) used by the program is logged in the log file.

HTTP API Commands

Command	Description
getver	Returns version of the program listening on the given port (such as "9.2.0"). **http://127.0.0.1:29592/getver**
getname	Returns the name of the program listening on the given port (such as "Macro Toolwo **http://127.0.0.1:29592/getname**
getvar	Returns the variable value. **http://127.0.0.1:29592/getvar?name=<VARIABLE_NAME>** • VARIABLE_NAME - a string that specifies an existing global application sco "ga_" prefix) or a system variable. It can be also an array element (for examp *Example: http://127.0.0.1:29592/getvar?name=_vClpText* *This request will retrieve the text currently stored in the clipboard.*
setvar	Sets the variable value. The request returns "1" on success or "0" if it fails. **http://127.0.0.1:29592/setvar?name=<VARIABLE_NAME>&value=<NEW_VAI** • VARIABLE_NAME - a string that specifies an existing global application sco "ga_" prefix) or a system variable. It can be also an array element (for examp • NEW_VALUE - a string that will be assigned as a value to the variable. *Example 1: http://127.0.0.1:29592/setvar?name=ga_Temperature&value="97"* *Example 2:* *http://127.0.0.1:29592/setvar?name=ga_CustomerName[0]&value="John J. Jame* *http://127.0.0.1:29592/setvar?name=ga_CustomerName[1]&value="Marion M. M*

runmacro	Runs an existing macro. Waits until the macro finishes and returns its result. **http://127.0.0.1:29592/runmacro?name=<MACRO_NAME>&param=<PARAM** • MACRO_NAME - a string that specifies a name of the existing macro. • PARAMETER - a string that will be passed to the macro as the parameter. *Example:* *Let's have a macro named "httpTest":* *<msg>(-100,-100,"httpTest: %_vMacroParameter%","",1,0,0,0)<#>* *<varset>("_vMacroResult=Goodbye","")* *Let's make this request:* *http://127.0.0.1:29592/runmacro?name=httpTest&param=Hello* *The macro will be called. It will display a message box with "Hello" text. When the request returns "Goodbye".*

The API supports UTF-8 encoding.

C# HTTP Client Code Example

```
namespace HttpClientCSharp
{
    class Program
    {
        static async Task Main(string[] args)
        {
            using var client = new HttpClient();
            var value = await
client.GetStringAsync("http://127.0.0.1:29592/getvar?name=ga_vMyVariable");

            Console.WriteLine(value);
        }
    }
}
```

Command Line Executable

The Macro Toolworks Professional edition (and Free Macro Player) comes with **MtwProxy.exe** command line executable. It provides other programs or scripts access to the variables and macros in running Macro Toolworks. Other programs can use MtwProxy.exe to set or retrieve a value of an application global variable (the one that starts with "ga_" prefix - see more here - or a system variable - see more here), and start an existing macro and retrieve its result.

MtwProxy.exe Command Line API Commands

Command	Description
getver	Writes version (such as "9.2.0") of the Macro Toolworks Professional edition (or Fr standard output. **MtwProxy.exe getver**
getname	Writes program name (such as Macro Toolworks) to the standard output. **MtwProxy.exe getname**
getvar	Retrieves a variable value and writes it to the standard output. **MtwProxy.exe getvar <VARIABLE_NAME>** • VARIABLE_NAME - a string that specifies an existing global application sco "ga_" prefix) or a system variable. It can be also an array element (for examp *Example: MtwProxy.exe getvar _vClpText* *This call will the text currently stored in the clipboard to the standard output.*
setvar	Sets the variable value and writes "1" (on success) or "0" (on fail) to the standard ou **MtwProxy.exe setvar <VARIABLE_NAME> <NEW_VALUE>** • VARIABLE_NAME - a string that specifies an existing global application sco "ga_" prefix) or a system variable. It can be also an array element (for examp • NEW_VALUE - a string that will be assigned as a value to the variable. *Example 1: MtwProxy.exe setvar ga_Temperature 97* *Example 2:* *MtwProxy.exe setvar ga_CustomerName[0] "John J. James"* *MtwProxy.exe setvar ga_CustomerName[1] "Marion M. Marr"*
runmacro	Runs an existing macro. Waits until the macro finishes and writes its result to the sta **MtwProxy.exe runmacro <MACRO_NAME> <PARAMETER>** • MACRO_NAME - a string that specifies a name of the existing macro. • PARAMETER - a string that will be passed to the macro as the parameter. *Example:* *Let's have a macro named "cmdTest":* *<msg>(-100,-100,"Http Test: %_vMacroParameter%","",1,0,0,0)<#>* *<varset>("_vMacroResult=Goodbye","")* *Let's run this command in Command Line window:* *MtwProxy.exe runmacro cmdTest Hello* *The macro will be called. It will display a message box with "Hello" text. When the "Goodbye" is written to the Command Line window.*

The data written to the standard output is in UTF-8 encoding. The executable returns 0 on success or -1 on error. You need to manually add path to the MtwProxy.exe to your "Path" environment variable if you want to use the MtwProxy.exe without the full path in scripts (as shown in the example below). The default path is c:\Program Files (x86)\Macro Toolworks\Bin (or c:\Program Files (x86)\Free Macro Player\Bin).

Example (.bat)

Let's have a macro named "cmdTest":

<msg>(-100,-100,"cmdTest: %_vMacroParameter%","",1,0,0,0)<#>
<varset>("_vMacroResult=Goodbye","")

Let's run this .bat file in Command Line window:

```
@echo OFF

for /F "delims=" %%a in ('MtwProxy.exe runmacro cmdTest "Hello, World!"') do set
result=%%a

IF %ERRORLEVEL% NEQ 0 (
   @echo FAIL
   EXIT /B
)

@echo %result%
@echo OK
```

Result: The macro will be called. It will display a message box with "Hello, World!" text. When the message box is closed text "Goodbye" is echoed in the Command Line window.

Windows Script (WScript)

The Macro Toolworks Professional edition (and Free Macro Player) comes with an **Mtw.Proxy** Windows Scripting object. It provides access to the variables and macros in running Macro Toolworks. JScript or VBScript scripts can use MtwProxy object to set or retrieve a value of an application global variable (the one that starts with "ga_" prefix - see more here - or a system variable - see more here), and start an existing macro and retrieve its result.

Mtw.Proxy Object API Commands

Command	Description
GetVer()	Returns version (such as "9.2.0") of the Macro Toolworks Professional edi **var mtwProxy = WScript.CreateObject("Mtw.Proxy");** **var version = mtwProxy.GetVer();**
GetName()	Returns program name (such as Macro Toolworks). **var mtwProxy = WScript.CreateObject("Mtw.Proxy");** **var name = mtwProxy.GetName();**
GetVar(VARIABLE_NAME)	Returns a variable value. **var mtwProxy = WScript.CreateObject("Mtw.Proxy");** **var value = mtwProxy.GetVar(VARIABLE_NAME);** • VARIABLE_NAME - a string that specifies an existing global applic "ga_" prefix) or a system variable. It can be also an array element (f
SetVar(VARIABLE_NAME, NEW_VALUE)	Sets the variable value and returns "1" (on success) or "0" (on fail). **var mtwProxy = WScript.CreateObject("Mtw.Proxy");** **var success = mtwProxy.SetVar(VARIABLE_NAME, NEW_VALUE);** • VARIABLE_NAME - a string that specifies an existing global applic "ga_" prefix) or a system variable. It can be also an array element (f • NEW_VALUE - a string that will be assigned as a value to the varial
RunMacro(MACRO_NAME, PARAMETER)	Runs an existing macro. Waits until the macro finishes and returns its result **var mtwProxy = WScript.CreateObject("Mtw.Proxy");** **var result = mtwProxy.RunMacro(MACRO_NAME, PARAMETER);** • MACRO_NAME - a string that specifies a name of the existing mac • PARAMETER - a string that will be passed to the macro as the para

The API parameters and return values are in the UTF-8 encoding. If you use a portable package (you download .zip package and not a .exe installer) then you need to manually register the MtwObj32.dll (32-bit) and MtwObj64.dll (64-bit) in order to make the Mtw.Proxy object work in WScript scripts. The dll's are by default located in c:\Program Files (x86)\Macro Toolworks\Bin (or c:\Program Files (x86)\Free Macro Player\Bin).

Example:

Let's have a macro named "wsTest":

<msg>(-100,-100,"wsTest: %_vMacroParameter%","",1,0,0,0)<#>

<varset>("_vMacroResult=Goodbye","")

Let's run this JScript:

```
function MtwProxySample()
{
.....................................................................
var WSHShell = WScript.CreateObject("WScript.Shell");
    .....................................................................
var Mtw = WScript.CreateObject("Mtw.Proxy");
.....................................................................
var result = Mtw.RunMacro("wsTest", "Hello");
.....................................................................
WSHShell.Popup(result);.....................................................
}

MtwProxySample();
```

The macro will be called. It will display a message box with "Hello" text. When the message box is closed text "Goodbye" is showed in the JScript pop up window.

Commands & Syntax

- Macro Syntax Basic
- Macro Command Syntax
- Macro Variables
- System Variables
- Expressions & Time Calculations
- Commands

General Macro Syntax Basics

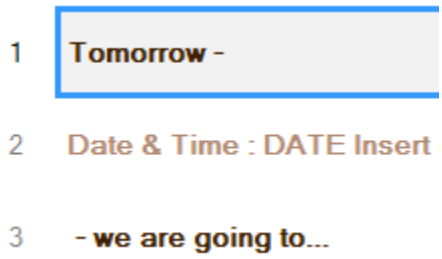
The simplest general macro is just a free text like "Hello World!". When the macro is played back, either (i) sequence of keystrokes representing this text is sent so that the text is "typed" to the target application (application that is active - i.e., receiving keyboard focus) or (ii) the text is placed to clipboard and is pasted to the active application by Ctrl+C hot-key shortcut (the active application must support such "paste" operation). It is configurable in macro as well as in program settings if keystrokes or clipboard is used. Such simple macros are used to quickly insert often used pieces of text (phrases, e-mail addresses, paragraphs, etc.) to different applications and documents. However, macros can also contain useful commands. If there is a command within the macro then it is automatically recognized and executed. Commands have its special syntax that looks like this:

<some_command>("param1", param2, ...)

An example of macro that combines a free text is as follows:

Tomorrow - *<date>(21,"/",1,1,0,"")* - we are going to...

The macro is presented and editable in this textual format if "Macro Text" is selected in macro editor. If "Macro Steps" is selected then the macro is presented and edited as a sequence of steps like this:



It depends on the Macro Toolworks usage - mostly used as text replacement utility or mostly used to write complex macros with many commands - what is the more appropriate way how to edit macros - as Macro Text or Macro Steps.

Note: There is a command <cmds> ("**Macro execution: Ignore free text, execute ONLY COMMANDS**") causing that only macro commands are executed and any other text (including new lines) is ignored.

Macro Command Syntax

There are commands without parameters and commands that can have parameters.

Commands Without Parameters
Macro commands without parameters have this syntax:

<command_without_parameters>

An example can be "*<clpempty>*" that clears clipboard content.
Note: Keyboard keys such as "Page Down" have also the same syntax (*<pgdn>*), however, if <cmds> command ("**Macro execution: Ignore free text, execute ONLY COMMANDS**") is in effect then the keys are not played back.

Commands With Parameters
Macro commands with parameters have this syntax:

<command_wit_parameters>("param1", param2, ...)

Command parameter can be:

1. **Static text (constant)**

2. **Variable**

3. **Combination** of a static text and variables. Variable used in a combination with static text must be enclosed in % chars ("a text %vVariable% text continues…). For example, to create a file with name based on the current date you can use this: <filecreate>("c:\myfiles\%_vCurrDate_Year%-%_vCurrDate_MM%-%_vCurrDate_DD%.doc",0). This will create c:\myfiles\2000-08-19.doc file.

4. **Expression**

Specific Parameters
There are commands that takes parameters that have specific syntax.

HWND	This type of parameter identifies an application window. Windows-spec as <actwin> takes this type of parameter. HWND is a unique handle W to identify each window. The HWND can be retrieved by some comma <win_enumerate>) or is provided by some system variables (_vKeybdFocusWindow_HWND, _vActiveWindow_HWND, _vActiveW While some other window attributes like window title or window class a can be other windows with the same title or class) the HWND is unique

Window Identifier Path (WIP)	Window Identification Path (WIP) is a sequence of information that allo Window. The WIP is used as a parameter for many windows command <win_findimage>, <win_captureimage>, <winclose>, etc. The user doe WIP manually when editing a window command in visual editor. The us cursor on the window he/she wants to use and the WIP is created autor contains a hierarchical structure (parent/child) of the windows. Each wi window title, window class and X, Y position (X,Y position is used for w only if there are multiple windows with the same window title and windo It is possible to use wildcards (* and ?) within a WIP. There are exampl window is activated: <actwin>("[A*.txt - Notepad|Notepad|#0|#0]",0,0,"no") - this command activates Notepad windows with titles such as "Audrey. "Aaron.txt - Notepad", "Abbey.txt - Notepad" <actwin>("[B*.txt - Notepad|Notepad|#0|#0]",0,0,"no") - this command activates Notepad windows with titles such as "Bred.tx "Boris.txt - Notepad", "Bradley.txt - Notepad" <actwin>("[*.txt - Notepad|Notepad|#0|#0]",0,0,"no") - this command activates Notepad windows with titles such as "Audrey. "Bradley.txt - Notepad", "Jane.txt - Notepad" <actwin>("[Untitled - Notepad|Notepad|#0|#0]",0,0,"no") - this command activates only Notepad window with title "Untitled - Note <actwin>("[*Notepad|Notepad|#0|#0]",0,0,"no") - this command activates Notepad windows with titles such as "Audrey. "Untitled - Notepad", "page.html - Notepad"
XmlDocumentHandle	This is a unique number that identifies an open Xml file. It is obtained a Xml related commands.
XmlElementHandle	This is a unique number that identifies an Xml document element. It is o some Xml related commands.
Workbook Identifier	This is a unique number that identifies an open Excel workbook. It is ob some Excel related commands.

Macro Variables

Variables

There are these types of variables:

- **Macro local variables** - The variable name can be any string (characters such +, -, \, /, <, >, *, :, (,), [,] cannot not be used) and the variable scope is only in a single macro. The macro variable is enclosed in **%** like in this example:
  Customer name: %varCustName%; Customer phone: %varCustPhone%
- **Global variables** - there are two types of global variables that differ by scope:
  - **macro** global variable - the variable is known in all the macros (called from each other using <run> command) during the macro execution. The macro global variable must begin with the *"**gm**_"* prefix (for example, "gm_varName").
  - **application** global variable - the variable is known during the whole application session (the value is remembered until the program is closed). The application global variable must begin with the *"**ga**_"* prefix (for example, "ga_varCurrentProject").
- **System variables** - the variables provide system dependent information and typically are read-only.
- **Procedure Parameters** - if a procedure parameter starts with "**par**" prefix (for example, "parInputText") then the parameter is local only within the procedure. Otherwise the parameter is either macro local or global as described above.
- **Local procedure variables** - if a variable defined within a procedure starts with "**lpv**" (**l**ocal **p**rocedure **v**ariable) prefix (for example, "lpvTemporaryVariable") then the variable is known and can be accessed only within the procedure.

Variable Array

Any variable (except system variables) can be used also as an array without any special declaration. For the array, this syntax is used:
varVariableName[index].

For the multiple dimension arrays this syntax is used:
varVariableName[index1:index2:index3:....:indexN].

Array variables do not have a value assigned automatically. Until the user assigns a value to a given variable (for example, using <varset>("vName[10]=John","") command) the variable value does not exist. This can be detected using "_vDoesVariableExist_VARNAME" system variable (_vDoesVariableExist_vName[10] from our example). The system variable "_vDoesVariableExist_VARNAME" contains 0 if the variable does not exist or 1 if it does.

Environment Variables

The program also recognizes Windows system environment variables such as %ProgramFiles% and such environment variables can be used in macros.

System Variables

System variables provide information that can be used in macro. System variables content is set by program automatically and cannot be changed.

List of system variables:
Window Related

_vActiveWindow	Title of the currently active window.
_vActiveWindow_HWND	HWND of the currently active window.
_vActiveWindowChild	Title of the currently active child window (dialog box, toolbox, do
_vActiveWindowChild_HWND	HWND of the currently active child window (dialog box, toolbox,
_vActiveWindowPrev	Title of the previously active window.
_vActiveWindowPrev_HWND	HWND of the previously active window .
_vWinFromMouseCur	Title of main window with mouse over.
_vWinFromMouseCur_HWND	HWND of main window with mouse over
_vChildWinFromMouseCur	Title of child window (dialog box, toolbox, document window) wit
_vChildWinFromMouseCur_HWND	HWND of child window (dialog box, toolbox, document window)
_vKeybdFocusWindow	Title of the window that is currently receiving keyboard input.
_vKeybdFocusWindow_HWND	HWND of the window that is currently receiving keyboard input.
_vKeybdFocusControl	Title of the window control that is currently receiving keyboard inp
_vKeybdFocusControl_HWND	HWND of the window control that is currently receiving keyboard
_vWinRectX1, _vWinRectY1, _vWinRectX2, _vWinRectY2, _vWinWdt, _vWinHgt, _vWinTitle, _vWinState, _vWinActive, _vWinClass , _vWinHWND	Attributes of the window retrieved using <wininfo> command.
_vMonitorCount	Number of monitors
_vMonitorPrimary	Primary monitor number (1 or higher).
_vMonitorWorkAreaX_*MNum*	Upper left X coordinate of a monitor work area. *(Replace MNu such as 1 - _vMonitorWorkX_1)*

_vMonitorWorkAreaY_*MNum*	Upper left Y coordinate of a monitor work area.
_vMonitorWorkAreaCX_*MNum*	Width of a monitor work area.
_vMonitorWorkAreaCY_*MNum*	Height of a monitor work area.
_vMonitorScreenX_*MNum*	Upper left X coordinate of a monitor screen.
_vMonitorScreenY_*MNum*	Upper left Y coordinate of a monitor screen.
_vMonitorScreenCX_*MNum*	Width of a monitor screen.
_vMonitorScreenCY_*MNum*	Height of a monitor screen.

Date & Time Related

_vCurrDate_DefaultShort	Windows default short date format.
_vCurrDate_DefaultLong	Windows default long date format.
_vCurrDate_Day	Day of the current date (e.g., 9 if the date is 02/09/2000).
_vCurrDate_DD	Day of the current date in the DD format (e.g., 09)
_vCurrDate_DayName	Name of the day of the current date (e.g., Sunday, Monday, etc.)
_vCurrDate_DayOfWeek	Day of the week (e.g, 1 for Sunday, 2 for Monday, …, 7 for Satur
_vCurrDate_Month	Month of the current date (e.g., 02 if the date is 02/19/2000).
_vCurrDate_MM	Month of the current date in the MM format (e.g., 02).
_vCurrDate_Year	Year of the current date (e.g., 2000 if the date is 02/19/2000).
_vCurrDate_YY	Year of the current date in the YY format (e.g., 00).
_vCurrDate_MonthName	Name of the month of the current date (e.g., February if the date
_vCurrDate_DDMMYYYY	Contains current date in DD/MM/YYYY form.
_vCurrDate_MMDDYYYY	Contains current date in MM/DD/YYYY form.
_vCurrTime_Hour12	Contains hour of the current time (e.g., 2 if the current time is 2:1
_vCurrTime_Hour24	Contains hour of the current time (e.g., 14 if the current time is 2:
_vCurrTime_Minute	Contains minute of the current time (e.g., 15 if the current time is
_vCurrTime_Second	Contains second of the current time (e.g., 0 if the current time is

_vCurrTime_AMPM	Contains PM/AM of the current time (e.g., PM if the current time
_vCurrTime_Total	Contains the number of milliseconds elapsed from the last reboo
_vCurrDateTime_ForCalc	The macro language natively supports time calculations, howeve be in appropriate format which is *"YYYY.MM.DD HH:MM:SS"* (This system variable contains current time in this format.

Files & Folders:

_vDirDateTime	Contains a string with current time and "Dir" prefix that can be us naming. *Example: "Dir_2007_11_24_~14~45~03"*
_vFileDateTimeTXT	Contains a string with current time and "File" prefix and ".txt" exte used for files naming. *Example: "File_2007_11_24_~14~45~03.txt"*
_vFileDateTimeDAT	Contains a string with current time and "File" prefix and ".dat" ex used for files naming. *Example: "File_2007_11_24_~14~45~03.dat"*
_vFolder_AppData	Contains path to the application data folder.
_vFolder_CommonDesktop	Contains path to the all users desktop folder.
_vFolder_CommonPrograms	Contains path to the all users start menu programs folder.
_vFolder_CommonStartMenu	Contains path to the all users start menu folder.
_vFolder_Cookies	Contains path to the cookies folder.
_vFolder_Desktop	Contains path to the desktop folder.
_vFolder_Favourites	Contains path to the favorites folder.
_vFolder_Personal	Contains path to the personal folder.
_vFolder_Programs	Contains path to the start menu programs folder.

_vFolder_Recent	Contains path to the recent files folder.
_vFolder_SendTo	Contains path to the "send to" folder.
_vFolder_StartMenu	Contains path to the start menu folder.
_vFolder_Startup	Contains path to the start menu start up folder.
_vFolder_Templates	Contains path to the templates folder.
_vFolder_Windows	Contains path to the windows folder.
_vFolder_System	Contains path to the system folder.
_vFolder_Temp	Contains path to the temporary folder.
_vFolder_AdminTools	Contains path to the admin tools folder.
_vFolder_BitBucket	Contains path to the recycle bin folder.
_vFolder_CDBurnArea	Contains path to the folder where files to be burned are stored.
_vFolder_CommonAdminTools	Contains path to the folder with admin tools for all users.
_vFolder_CommonAltStartup	Contains path to the folder corresponding with nonlocalized Star all users.
_vFolder_CommonDocuments	Contains path to the documents folder for all users.
_vFolder_CommonMusic	Contains path to the music folder for all users.
_vFolder_CommonPictures	Contains path to the pictures folder fro all users.
_vFolder_CommonTemplates	Contains path to the templates folder for all users.
_vFolder_CommonVideo	Contains path to the videos folder for all users.
_vFolder_Fonts	Contains path to the fonts folder.
_vFolder_InternetCache	Contains path to the IE cache folder.
_vFolder_MyDocuments	Contains path to the documents folder for logged user.

_vFolder_MyMusic	Contains path to the music folder for logged user.
_vFolder_MyPictures	Contains path to the pictures folder for logged user.
_vFolder_MyVideo	Contains path to the videos folder for logged user.
_vFolder_Resources	Contains path to the resource data folder.
_vReportDateTimeTXT	Contains a string with current time and "Report" prefix and ".txt" used for files naming. *Example: "Report_2007_11_24_~14~45~03.dat"*

Drag & Drop:

The variables described here are only accessible after user dr toolbar button. (Macro toolbars are only available in MacroToo Toolworks products.) The variables allow to further process (de etc.) the file(s) dropped.

_vDropFile_FileFullPath	Full path to the file dropped.
_vDropFile_FileName	File name of the file dropped.
_vDropFile_FileDir	Directory of the file dropped.
_vDropFile_Num	The number of files dropped

Special Characters

_vCmdDelayKeyMs	Delay in milliseconds after each keystroke in macro. The default Setting this variable allows to slow down execution of keystrokes
_vKeyReturn	'Return' (new line) key (use this key to add a new line to a text in equivalent to \r\n codes.
_vKeyNewLine	Contains equivalent to \n code (new line).
_vKeyCR	Contains equivalent to \r code (carriage return).
_vKeyTab	'Tab' key (use this key to add 'tab' to a text in variable).
_vQuoteChar	" character (use this key to add " to a text).

_vKeySpace	“space” character
_vKeyPercent	% character
_vKeyBigger	> character
_vKeySmaller	< character
_vKeyBracketL	(character
_vKeyBracketR	) character
_vKeySqBracketL	[character
_vKeySqBracketR	] character
_vKeyComma	, character

Mouse & Keyboard Related

_vCmdDelayMouseMs	Delay in milliseconds after each mouse event (such as mouse bu move, mouse double-click, etc.) in macro. The default value is 5. allows to slow down (speed up) execution of mouse events.
_vLastMouseClick	Contains last mouse click event in the <waitfor> command: <mlbu> - left button click <mmbu> - middle button click <mrbu> - right button click
_vLastKey	Contains last key pressed in the <waitfor> command.
_vMousePosX	Mouse cursor position in the screen coordinates - X.
_vMousePosY	Mouse cursor position in the screen coordinates - Y.
_vCursorPosX	X-position (in absolute coordinates) of blinking cursor that is sho text. If the cursor is not present or its position cannot be retrieved
_vCursorPosY	Y-position (in absolute coordinates) of blinking cursor that is sho text. If the cursor is not present or its position cannot be retrieved
_vIsCapsLockON	Is set to YES if CapsLock is on. Otherwise the variable contains
_vIsNumLockON	Is set to YES if NumLock is on. Otherwise the variable contains
_vIsScrollLockON	Is set to YES if ScrollLock is on. Otherwise the variable contains

_vMouseCursorShape	Current shape of the mouse cursor. Can be one of these: ARROW BEAM WAIT CROSS UPARROW SIZENWSE SIZENESW SIZEWE SIZENS SIZEALL NO HAND APPSTARTING HELP

Clipboard Related

_vClpFormats	Contains comma separated list of formats of data currently save
_vClpSize	Contains size of clipboard data.
_vClpText	Contains text clipboard data (if currently available).
_vClpSequenceNumber	Contains clipboard sequence number that is increased any time is changed.
_vClpTextHistoryEnable	If this variable is set to "YES" (must be set by a macro) then the l _vClpText1, etc. contains historical clipboard data.
_vClpText0, _vClpText1,...,_vClpText9	Contains historical clipboard textual data. _vClpText0 contains tl _vClpText9 contains the oldest data. The content of the _vClpTe variables change always when new text data is copied to clipboa data is only collected if _vClpTextHistoryEnable is set to "YES".

Macro Flow Related

_vCanceled	The variable is set to 1 if a command (<form_show>, <extmacro <var_oper>) si canceled by user. Otherwise the variable contai
_vLoopCounter	Loop counter (see <begloop> and <endloop> commands). Start
_vLoopCounter0	Loop counter (see <begloop> and <endloop> commands). Start
_vIsConnectedToInternet	If the computer is connected to the Internet the variable value is e otherwise it is NO.
_vErr	If an error occurs during macro execution, the error description is variable.

_vIsExecOnlyCommands	After <cmds> command is executed, this variable content is YES <keys> command is executed it is NO.
_vWinStarting	If the program is set to automatically start on windows startup the contains "YES" when the program is starting on windows startup is "NO".
_vMacroParameter	This variable contains parameter that is passed to macro started commands: <run> <extmacro> <remote_macro_call> [Note: Unlike other system variables, the value of this variable c <varset> command.]
_vMacroResult	This variable contains result returned from macro started using f <run> <extmacro> <remote_macro_call> [Note: Unlike other system variables, the value of this variable c <varset> command.]
_vMacroFileLoaded	Full path to the macro file currently loaded in the program. It is also possible to write to this variable in order to force the pro macro file: <varset>("_vMacroFileLoaded=c:\....\otherfile.4tw","") *Note: The other file is loaded when there is no macro currently*
_vMacroSharedFileLoaded	Full path to the shared macro file currently loaded in the program
_vMacroFileFolder	Folder where the currently loaded macro file resides. It is ended
_vRunningMacroName	Contains name of the currently running macro.
_vRunningMacroId	Contains Id of the currently running macro.
_vMsgButton	The variable contains YES if "Yes" button was clicked in the <ms otherwise it contains NO.

_vIsMacroEnabled_*MACRONAME*	Such system variable exist for each macro. The variable contai enabled, otherwise it is "0". For example, if the macro name is " associated system variable is "_vIsMacro_Enabled_MyMacro1"
_vIsGroupEnabled_*MACRONAME*	Such system variable exist for each macro group. The variable macro group is enabled, otherwise it is "0". For example, if the r is "FileMacros" then the associated system variable is "_vIsGroupEnabled_MyMacro1".
_vDoesVariableExist_*VARIABLE*	Such system variable exist for each variable. The variable conta exists, otherwise it is "0". For example, if the name of a variable the associated system variable is "_vDoesVariableExist_vMyDa
_vWaitForImage_PosX	This variable contains X position of the image that was found on <waitfor>(...IMAGE_ON_SCREEN...) command.
_vWaitForImage_PosY	This variable contains Y position of the image that was found on <waitfor>(...IMAGE_ON_SCREEN...) command.

Power Status (laptop)

_vPowerAcStatus	This variable is: • "1" if there is a power connected to the laptop • "0" if the power is is not connected to the laptop • "unknown" if the status is unknown (not laptop)
_vPowerBatteryLifePercent	This variable contains either the percentage of battery charge re range 0 to 100) or "unknown" if status is unknown (not laptop).
_vPowerBatteryLifeSeconds	This variable contains either the number of seconds of the batter or "unknown" if status is unknown (not laptop).

Other:

_vLastWebPageLoaded	This variable contains URL of the web page last opened using < <www_fillform>..
_vLastAppExitCode	Exit code of last application that was executed by <execappex> parameter "wait for exited".
_vLastExecutedFileProcessId	Process Id of the last file opened using <execappex> command.
_vOS_UserDefaultLanguageID	Contains the language identifier of the current Windows user loc
_vOS_SystemDefaultLanguageID	Contains the language identifier of the Windows system locale.
_vStrEmpty	Empty string (use this variable to assign empty string to a variab

_vIsScreensaverRunning	YES if screen saver is running.
_vIsScreensaverEnabled	YES if screen saver is enabled.
_vOperatingSystemVersion	Operating system version.
_vScreenWidth	Screen width in pixels.
_vScreenHeight	Screen height in pixels.
_vThisProgramVersion	Contains #THIS_PROGRAM# internal version.

Expressions & Time Calculations

Expressions can be used as command parameters including "if" command condition. Expressions have this syntax:

EXPR(expression)

or

EXPRXX(expression)

Note: XX specifies how many digits follow after decimal point (real numbers precision) if the expression calculates in real numbers.

Expressions can be used as a parameter for almost all commands. Expressions support all basic mathematical operations and brackets.

Example1: <mm>(EXPR(%_vScreenWidth%/2), EXPR(%_vScreenHeight%/2))
Example2: <varset>("vPercent=EXPR(100(%vItems%/%vItemsTotal%))","")*

The expressions support also **date/time arithmetic's.** The date/time must be in ISO format which is "**yyyy-mm-ddThh:mi:ss**", for example 2018-04-16T21:05:58. The difference between two times is "time span" that has this format "**days hh:mi:ss**", for example 4 02:0:01 (four days and two hours and one second difference). The time span is also used to add/subtract amount of time to/from given time. This operations are supported:

"days hh:mi:ss" = "yyyy-mm-ddThh:mi:ss" - "yyyy-mm-ddThh:mi:ss"
"yyyy-mm-ddThh:mi:ss" = "yyyy-mm-ddThh:mi:ss" + "days hh:mi:ss"
"yyyy-mm-ddThh:mi:ss" = "yyyy-mm-ddThh:mi:ss" - "days hh:mi:ss"
"days hh:mi:ss" = "days hh:mi:ss"+"days hh:mi:ss"
"days hh:mi:ss" = "days hh:mi:ss"-"days hh:mi:ss"

Example3: <varset>("vTomorrowDate=EXPR(%_vCurrDateTime_ISO%+1 0:0:0)","") <#> 1 day later (next day)
Example4: <varset>("vDate=EXPR(%_vCurrDateTime_ISO%-10 13:45:10)","") <#> What date was before 10 days, 13 hours, 45 minutes and 10 seconds
Example5: <varset>("vTimeSpan=EXPR(%_vCurrDateTime_ISO%-2005.11.21 13:45:00)","") <#> Howmuch time left since 2005/11/21, 13:45:00

There are these **limitations** on time input: 1970 <= yyyy<= 3000; 1 <= mm <= 12; 1 <= dd <= 31; 0 <= hh <= 23; 0 <= mi <= 59; 0 <= ss <= 59

Commands

Available commands are logically organized in groups. Each command comes with a simple example showing how to use the command in a macro.

Free Text

- ... [Free]

A free macro text

Available in: Free edition

When macro is started, the macro free text is sent to currently active application (window). The macro free text is sent to the active application either as a sequence of keystrokes or pasted through clipboard - what option is used depends on the program settings and the macro settings.

The macro free text is typically used in combination with "text shortcut" keyboard trigger to implement simple "text replacement" macros.

The macro free text can be combined with any macro command - just add macro command anywhere in the macro text.

Example (Macro Steps):

1 **If this macro is run in Notepad (for example) then whole this text is inserted to Notepad.**

Example (Plain Text):

If this macro is run in Notepad (for example) then whole this text is inserted to Notepad.

Clipboard

SAVE - < clpsave >() ... [Pro]

Clipboard SAVE
<clpsave>("File path")
Available in: Professional edition

Saves clipboard content to file. Depending on the file extension the clipboard content is saved different ways:
.clx - whole the clipboard content (all clipboard formats) is saved
.txt - just plain text - if available in clipboard - is saved to file
.bmp, .jpg, .png, .gif - just picture - if available in clipboard - is saved to file

#	Parameter name	Parameter description
1	File path	Full path to the file that receives clipboard data.

Example (Macro Steps):

1 <#> <#> This example saves clipboard data to file c:\clpdata.clx

2 Macro execution: ONLY COMMANDS

3 Clipboard SAVE File path=c:\clpdata.clx

Example (Plain Text):

```
<#> This example saves clipboard data to file c:\clpdata.clx
<#>
<cmds>
<clpsave>("c:\clpdata.clx")
```

LOAD - < clpload >() ... [Pro]

Clipboard LOAD
<clpload>("File path")
Available in: Professional edition

Loads clipboard content from file. It can be a file previously created using "clipboard save" command or by "Tools\Save Clipboard Data To File" menu command. It can be also a .txt file or a .bmp, .png, .jpg, .gif image file. This command changes clipboard content.

#	Parameter name	Parameter description
1	File path	Full path to file.

Example (Macro Steps):

```
1   <#> <#> This example loads clipboard data from file "c:\clpdata.clx"
2   Macro execution: ONLY COMMANDS
3   IF FILE "c:\clpdata.clx" Exist ()
4       Clipboard LOAD File path=c:\clpdata.clx
5   ELSE activate
6       Message SHOW "" : "The file 'c:\clpdata.clx' doesn't exist." (other parameters: x = 100, y = 100, Window
        title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
7   ENDIF
```

Example (Plain Text):

```
<#> This example loads clipboard data from file "c:\clpdata.clx"
<#>
<cmds>

<if_file>("c:\clpdata.clx","EXIST","")
  <clpload>("c:\clpdata.clx")
<else>
  <msg>(100,100,"The file 'c:\clpdata.clx' doesn't exist.","Message",1)
<endif>
```

PASTE - < clppastetext >() ... [Free]

Clipboard PASTE
<clppastetext>("Text to paste")
Available in: Free edition

The command inserts text to active application (window) through clipboard simulating the paste function. The command puts text passed as the command parameter to the clipboard and then invokes "Ctrl+V" in the active application.

#	Parameter name	Parameter description
1	Text to paste	Text (variable) to be pasted to destination application (document).

Example (Macro Steps):

```
1   <#> <#> This macro pastes "Hello!" into the Notepad
2   Macro execution: ONLY COMMANDS
3   IF WINDOW "[* - Notepad|Notepad|#46|#118]" Is Open (Match=Partial)
4       Window ACTIVATE bring "[* - Notepad|Notepad|#46|#118]" window to top (other parameters: Match =
        Partial, Window state = Normal, %p4_name = )
5       Clipboard PASTE Text to paste=Hello!
6   ELSE activate
7       Message SHOW "" : "'Notepad' is not opened!" (other parameters: x = 100, y = 100, Window title =
        Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
8   ENDIF
```

Example (Plain Text):

```
<#> This macro pastes "Hello!" into the Notepad
<cmds>

<if_win>("[* - Notepad|Notepad|#46|#118]","OPEN",0)
  <actwin>("[* - Notepad|Notepad|#46|#118]",0,0)
  <clppastetext>("Hello!")
<else>
  <msg>(100,100,"'Notepad' is not opened!","Message",1)
<endif>
```

CLEAR - < clpempty > ... [Pro]

Clipboard CLEAR
<clpempty>
Available in: Professional edition

This command clears the clipboard content.

Example (Macro Steps):

1	**<#> <#> This macro clears clipboard content**
2	**Clipboard CLEAR**

Example (Plain Text):

```
<#> This macro clears clipboard content
<#>
<clpempty>
```

COPY - < clpput >() ... [Pro]

Clipboard COPY
<clpput>("Text or file shortcut to copy:")
Available in: Professional edition

Puts text data or file shortcut to clipboard. This command changes clipboard content.

#	Parameter name	Parameter description
1	Text or file shortcut to copy:	Text to be put into the clipboard. It is also possible to put a file shortcut to the clipboard this way: FILE:file path. Example: ("FILE:c:\temp\file.txt") command puts shortcut to "c:\temp\file.txt" file to clipboard so that the file then can be copied using Windows Explorer "Paste" menu command.

Example (Macro Steps):

1 <#> <#> This macro puts data you type to clipboard

2 **Macro execution: ONLY COMMANDS**

3 **Variable SET** "vData=", Message text="Type text you want to put to clipboard"

4 **Clipboard COPY** "vData"

Example (Plain Text):

```
<#> This macro puts data you type to clipboard
<#>
<cmds>
<varset>("vData=","Type text you want to put to clipboard")
<clpput>("vData")
```

COPY SELECTED - < clp_copyselected >() ... [Pro]

Clipboard COPY SELECTED
<clp_copyselected>(Copy hotkey used,Timeout (seconds))
Available in: Professional edition

Copies selected data (text in a word processor, graphics in a graphics program, file in Windows Explorer, etc.) to clipboard. This command mimics pressing Ctrl+C (or Ctrl+Insert) key combination.

#	Parameter name	Parameter description
1	Copy hotkey used	0 - Ctrl+C key combination is used to copy the data to clipboard. 1 - Ctrl+Insert key combination is used to copy the data to clipboard.
2	Timeout (seconds)	

Example (Macro Steps):

1 <#> <#> This example copies selected data to clipboard

2 **Macro execution: ONLY COMMANDS**

3 **Error message DISABLED**

4 **Clipboard COPY SELECTED** Copy hotkey used=Ctrl+C, Timeout (seconds)=

5 **IF STRING** _vErr != NO

6 **Message SHOW** "" : "Cannot save data to clipboard - no text (or other object) is selected." (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

7 **ENDIF**

Example (Plain Text):

```
<#> This example copies selected data to clipboard
<#>
<cmds>
<me_error_nodisplay>
<clp_copyselected>(0)
<if_str>("_vErr != NO")
   <msg>(-100,-100,"Cannot save data to clipboard - no text (or other object) is selected.","Message",1)
<endif>
```

Replace text - < clp_replace_text >() ... [Pro]

Clipboard Replace text
<clp_replace_text>("Old text","New text",What,Method)
Available in: Professional edition

This command modifies current clipboard content (all textual formats) by replacing or inserting a portion of the text in the clipboard. It is possible to replace string or matching wildcard or matching regular expression by other string.

#	Parameter name	Parameter description
1	Old text	Current text in the clipboard to match.
2	New text	New text to be inserted to the clipboard.
3	What	Tells whether plain text, wildcard or regular expression should be used to match the current text in the clipboard. The values are "text", "wildcard" or "regex".
4	Method	Tells how to insert new text. Options are: replace - The matching text in the clipboard is replaced by new text insert_before - New text is inserted right in front of the matching text insert_after - New text is inserted next to the matching text

Example (Macro Steps):

1 <#> <#>This macro shows how to replace xxx text in clipboard by current date

2 **Clipboard COPY** "Put date here: xxx"

3 **Date & Time : DATE Insert or save to Variable** Format=Windows user default - long, Separator=/, Day leading zero=Yes, Month leading zero = Yes, Day shift = 0, Variable for result = vDate

4 **Clipboard Replace text** Old text = "xxx", New text = "%vDate%", What = "Text", Method = "Replace text"

Example (Plain Text):

```
<#>This macro shows how to replace xxx text in clipboard by current date
<clpput>("Put date here: xxx")<#>
<date>(21,"/",1,1,0,"vDate")<#>
<clp_replace_text>("xxx","%vDate%",text,replace)
```

Comments

Comment Line - < # > ... [Free]

<#> Comment Line
<#>
Available in: Free edition

This command has a single line comment meaning. The text on this line is ignored when macro is executed.

Example (Macro Steps):

1	**<#> <#> This macro will do nothing...**
2	**<#> <#>... because it contains...**
3	**<#> <#>... only comments.**

Example (Plain Text):

<#> This macro will do nothing...
<#>... because it contains...
<#>... only comments.

Comment Block BEGIN { - < {# > ... [Pro]

<#> Comment Block BEGIN {
<{#>
Available in: Professional edition

This command begins a comment block. All macro text (macro steps) that are enclosed between <{#> and <}#> are ignored during macro execution.

Example (Macro Steps):

1	**<#> <#>This macro does nothing because everything is commented out**
2	**<#> Comment Block BEGIN {**
3	**Keyboard ScrollLock ON**
4	**Everything is ignored....**
5	**<#> Comment Block END }**

Example (Plain Text):

<#>This macro does nothing because everything is commented out
<{#><ScrollLock_ON>Everything is ignored....<}#>

Comment Block END } - < }# > ... [Pro]

<#> Comment Block END }
<}#>
Available in: Professional edition

This command ends a comment block. All macro text (macro steps) that are enclosed between <{#> and <}#> are ignored during macro execution.

Example (Macro Steps):

1	**<#> <#>This macro does nothing because everything is commented out**
2	**<#> Comment Block BEGIN {**
3	**Keyboard ScrollLock ON**
4	**Everything is ignored....**
5	**<#> Comment Block END }**

Example (Plain Text):

<#>This macro does nothing because everything is commented out
<{#><ScrollLock_ON>Everything is ignored....<}#>

Date & Time

: DATE Insert or save to Variable - < date >() ... [Free]

Date & Time : DATE Insert or save to Variable
<date>(Format,"Separator",Day leading zero,Month leading zero,Day shift,"Variable for result","Date")
Available in: Free edition

The command inserts date information (e.g., 02/24/2000) in required format to the currently active window or saves it to required variable. The command uses current date and time unless other date/time is supplied as the last parameter.

#	Parameter name	Parameter description
1	Format	The date format. There are several formats predefined and can be referenced by number: 0 = DDMMYYYY 1 = MMDDYYYY 2 = DD 3 = MM 4 = DDMM 5 = MMDD 6 = MMYYYY 7 = DD_January_YYYY 8 = January_DD_YYYY 9 = Friday_X_January_DD_X_YYYY 10 = Friday_January_DD_X_YYYY 11 = Friday_DD_January 12 = Friday_DD_January_YYYY 13 = Friday 14 = January 15 = YYYY 16 = YY 17 = YYYYMMDD 18 = YYYYDDMM 19 = YYYYMM 20 = Windows user default - short 21 = Windows user default - long 22 = Format for calculations and compare It is possible to define the format using a free form text with placeholders (for example, DD/MM/YY). Th eplaceholders are: DD - the day (such as 23) DN - the day name (such as Monday) DW - the day of the week (such as 1) MM - the month (such as 2) MN - the month name (such as February) YYYY - the year (such as 2019) YY - two digits year (such as 19 for 2019) Note: The format cannot contain comma character and brackets. Use a system variable such as %_vKeyBracketL%.
2	Separator	Separator character: .,/- etc.
3	Day leading zero	If 1, the day is displayed with leading zero (e.g., 02). If 0, the day is displayed without leading zero (e.g., 2).
4	Month leading zero	If 1, the month is displayed with leading zero (e.g., 09). If 0, the month is displayed without leading zero (e.g., 9).
5	Day shift	The date shift in days relative to the current date. 0 for no shift (today's date), -1 for date one day back (yesterday), +1 for date one day later (tomorrow), etc.
6	Variable for result	Variable that receives the date. If this variable name is an empty string, the date is send to active application as a set of keystrokes.
7	Date	Date input - the date to be used instead of the current date. It must be in ISO format (yyyy-mm-ddThh:mm:ss).

Example (Macro Steps):

```
1   <#> <#> Run this example in Notepad:
2   Current date is:
3   Date & Time : DATE Insert or save to Variable Format=MMDDYYYY, Separator=/, Day leading zero=Yes,
    Month leading zero = Yes, Day shift = 0, Variable for result =
4   \ Current date is:
5   Date & Time : DATE Insert or save to Variable Format=Friday January DD X YYYY, Separator=/, Day
    leading zero=Yes, Month leading zero = Yes, Day shift = 0, Variable for result =
6   \ Yesterday:
7   Date & Time : DATE Insert or save to Variable Format=MMDDYYYY, Separator=/, Day leading zero=Yes,
    Month leading zero = Yes, Day shift = -1, Variable for result =
8   \ 2020-11-23 in different format:
9   Date & Time : DATE Insert or save to Variable Format=Windows user default - long, Separator=/, Day
    leading zero=Yes, Month leading zero = Yes, Day shift = 0, Variable for result =
```

Example (Plain Text):

```
<#> Run this example in Notepad:
Current date is: <date>(1,"/",1,1,0,"")
Current date is: <date>(10,"/",1,1,0,"")
Yesterday: <date>(1,"/",1,1,-1,"")
2020-11-23 in different format: <date>(21,"/",1,1,0,"","2020-11-23")
```

: TIME Insert or save to Variable - < time >() ... [Free]

Date & Time : TIME Insert or save to Variable
<time>(Minute shift,Hour leading zero,Minute leading zero,Format,Time base,"Variable for result")
Available in: Free edition

The command inserts time (e.g., 2:09 AM) into the currently active window or saves it to required variable.

#	Parameter name	Parameter description
1	Minute shift	The time shift in minutes relative to the current time. 0 for no shift, -1 for time one minute back, +1 for time one minute later, etc. Can be a number or variable containing numeric value.
2	Hour leading zero	If 1, the hour is displayed with leading zero (e.g., 02). If 0, the hour is displayed without leading zero (e.g., 2). Can be a number or variable containing numeric value.
3	Minute leading zero	If 1, the minute is displayed with leading zero (e.g., 09). If 0, the minute is displayed without leading zero (e.g., 9). Can be a number or variable containing numeric value.
4	Format	Format used to produce time information. This is a free form text that can contain the following placeholders: HH - is replaced by hours MM - is replaced by minutes SS - is replaced by seconds a.m.p.m. - is replaced by a.m. or p.m. ampm - is replaced by am or pm A.M.P.M. - is replaced by A.M. or P.M. AMPM - is replaced by AM or PM Default format is HH:MM:SS. Note: The format cannot contain comma character and brackets. Use a system variable such as %_vKeyBracketL%.
5	Time base	If 0, the 24 hours time base is used. If 1, the 12 hours time base is used.
6	Variable for result	Variable that receives the time. If this variable name is an empty string, the time is send to active application as a set of keystrokes.

Example (Macro Steps):

1 <#> <#> Run this macro in Notepad:

2 **Current time (24-hours base, no AM/PM) is:**

3 **Date & Time : TIME Insert or save to Variable** Minute shift=0, Hour leading zero=Yes, Minute leading zero=Yes, Format = 0, Time base = 24 Hours, Variable for result =

4 **\ Current time (12-hours base, with AM/PM) is:**

5 **Date & Time : TIME Insert or save to Variable** Minute shift=0, Hour leading zero=Yes, Minute leading zero=Yes, Format = 1, Time base = 12 Hours, Variable for result =

Example (Plain Text):

<#> Run this macro in Notepad:

Current time (24-hours base, no AM/PM) is: <time>(0,1,1,0,0,"")
Current time (12-hours base, with AM/PM) is: <time>(0,1,1,1,1,"")

Display / Computer Screen

GET PIXEL - < display_getpixel >() ... [Pro]

Display GET PIXEL
<display_getpixel>(x,y,Variable)
Available in: Professional edition

Retrieves color on the given position on screen.

#	Parameter name	Parameter description
1	x	X-coordinate (in screen coordinates) of the point.
2	y	Y-coordinate (in screen coordinates) of the point.
3	Variable	Variable that receives the color (as a number).

Example (Macro Steps):

```
1   <#> <#> This macro will show how to use 'get pixel' command:
2   Macro execution: ONLY COMMANDS
3   Message SHOW "" : "Move mouse cursor to white color area and press 'Enter' key." (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
4   Display GET PIXEL color from position [ x=_vMousePosX, y=_vMousePosY ] to variable "vColor"
5   IF NUMERIC vColor==16777215
6       Message SHOW "" : "Yes, mouse cursor is on white color area." (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
7   ELSE activate
8       Message SHOW "" : "No, mouse cursor is not on white color area." (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
9   ENDIF
```

Example (Plain Text):

```
<#> This macro will show how to use 'get pixel' command:
<#>
<cmds>

<msg>(-100,-100,"Move mouse cursor to white color area and press 'Enter' key.","Message",1)

<display_getpixel>(_vMousePosX,_vMousePosY,vColor)

<if_num>("vColor==16777215")
  <msg>(-100,-100,"Yes, mouse cursor is on white color area.","Message",1)
<else>
  <msg>(-100,-100,"No, mouse cursor is not on white color area.","Message",1)
<endif>
```

CHANGE WALLPAPER - < display_changewallpaper >() ... [Pro]

Display CHANGE WALLPAPER
<display_changewallpaper>(File)
Available in: Professional edition

This command changes wallpaper on Windows desktop.

#	Parameter name	Parameter description
1	File	Full path to file with wallpaper image.

Example (Macro Steps):

```
1   <#> <#> This macro changes wallpaper image
2   Macro execution: ONLY COMMANDS
3   Variable OPERATION "SELECT_FILE" (Variable for result = vWallpaperFile, Input text/variable = c:\winnt\*.bmp, Parameter 1 = , Parameter 2 = , Parameter 3 = 0)
4   IF STRING vWallpaperFile != _vEmptyStr
5       Display CHANGE WALLPAPER "vWallpaperFile"
6       Message SHOW "" : "Wallpaper changed!" (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
7   ENDIF
```

Example (Plain Text):

```
<#> This macro changes wallpaper image
<#>
<cmds>

<var_oper>(vWallpaperFile,"c:\winnt\*.bmp",SELECT_FILE,"","", "0")

<if_str>("vWallpaperFile != _vEmptyStr")
  <display_changewallpaper>(vWallpaperFile)
  <msg>(-100,-100,"Wallpaper changed!","Message",1)
<endif>
```

Image FIND on SCREEN - < display_findimage >() ... [Pro]

Display Image FIND on SCREEN

<display_findimage>("Image file",Start search X,Start search Y,Variable for image found X,Variable for image found Y,Image match,Search area width,Search area height)
Available in: Professional edition

This command searches for defined image(s) on computer screen. If the image is found on the screen the command sets supplied coordinate variables. If multiple image files are defined using wildcards then [x,y] position variables for each image are created for each image file. For example, if image files are defined using "c:\find_images*.bmp" and there are "Button.bmp" and "Title.bmp" image files in the "c:\find_images" folder then variables "button.bmp_x" and "button.bmp_y" variables that defines position of the "Button.bmp" image on the screen is created. The same for "title.bmp" file there are "title.bmp_x" and "title.bmp_y" variable created. Using such variables it is possible to determine what images were found and what are their position on the screen. Image file names are always converted to lowercase.

Important:

The bitmap file the command is finding must be captured with the same DPI (or zoom in web browser) as the content presented on the screen. These are typical problems why the command fails:
1. The bitmap file is captured on a monitor with higher/lower DPI than the monitor where the command is finding the image. To prevent this problem always capture the image on the same monitor where the command is executed.

2. The bitmap file is captured in web browser with different zoom setting than the current zoom. To prevent this use the same web browser and the same zoom setings when capturing image and running the macro.

#	Parameter name	Parameter description
1	Image file	Full path to the image file. This is a bitmap image that is captured using "Capture..." feature in the command editor. In addition, multiple image files are supported by wildcards (* and ?).
2	Start search X	X-coordinate where to start searching on the computer screen.
3	Start search Y	Y-coordinate where to start searching on the computer screen.
4	Variable for image found X	Name of the variable that receives X-coordinate of the position of the image on the screen. If the image is not found then the variable receives "-1".
5	Variable for image found Y	Name of the variable that receives Y-coordinate of the position of the image on the screen. If the image is not found then the variable receives "-1".
6	Image match	If 0 then the image does not has to exactly match, a certain level of tolerance is allowed. If 1 then the image has to exactly match.
7	Search area width	It is possible to scope searching to an area smaller than the whole screen. This attribute specifies the width of the searching area. If this parameter is set to "0" then the whole screen width is being searched.
8	Search area height	It is possible to scope searching to an area smaller than the whole screen. This attribute specifies the height of the searching area. If this parameter is set to "0" then the whole screen height is being searched.

Example (Macro Steps):

```
1   <#> <#> This example shows how to find "Start" button on the screen and click on it

2   Macro execution: ONLY COMMANDS

3   Display Image FIND on SCREEN "C:\PicturesForMacros\StartButton.bmp" (Start search X = 0, Start search
    Y = 0, Variable for image found X = vStartButtonX, Variable for image found Y = vStartButtonY, Image match =
    0, Search area width = , Search area height = )

4   IF %vStartButtonX% > -1

5       <#> <#> Start button found, move cursor on it...

6       Mouse MOVE position [ x=%vStartButtonX%, y=%vStartButtonY% ]

7       <#> <#> ...and click

8       Mouse BUTTON: LEFT button DOWN

9       Mouse BUTTON: LEFT button UP

10  ELSE activate

11      <#> <#> Start button not found, show message.

12      Message SHOW "Error" : "The Start button was not find on the screen. Possible reasons are: 1) The Task
        bar is hidden. 2) The Start button is overlapped by some window. 3) The StartButton.bmp file contains
        picture other than how Start button looks like." (other parameters: x = -100, y = -100, Window title =
        Message, Buttons = OK, Timeout (seconds) = 0, Always on top = ).

13  ENDIF
```

Example (Plain Text):

```
<#> This example shows how to find "Start" button on the screen and click on it
<cmds>

<display_findimage>("C:\PicturesForMacros\StartButton.bmp",0,0,vStartButtonX,vStartButtonY, 0)

<if>("%vStartButtonX% > -1")

  <#> Start button found, move cursor on it...
  <mm>(%vStartButtonX%,%vStartButtonY%)

  <#> ...and click
  <mlbd><mlbu>

<else>

<#> Start button not found, show message.
<msg>(-100,-100,"The Start button was not find on the screen. Possible reasons are:

1) The Task bar is hidden.
2) The Start button is overlapped by some window.
3) The StartButton.bmp file contains picture other than how Start button looks like.","Message",1,0,2)

<endif>
```

Image CAPTURE from SCREEN - < display_captureimage >() ... [Pro]

Display Image CAPTURE from SCREEN
<display_captureimage>(x,y,Width,Height,"Image file")
Available in: Professional edition

This command captures an image on computer screen.

#	Parameter name	Parameter description
1	x	X coordination of the upper left corner of the area to be captured.
2	y	Y coordination of the upper left corner of the area to be captured.
3	Width	Width of the area to be captured. If it is set to "0" then whole screen is captured.
4	Height	Height of the area to be captured.
5	Image file	Name (or full path) of the resulting image file.

Example (Macro Steps):

1 <#> <#> This macro shows how to capture display picture

2 **Macro execution: ONLY COMMANDS**

3 **Display Image CAPTURE from SCREEN** x = "0", y = "0", Width = "0", Height = "0", Image file = "%_vFolder_Personal%\ScreenImage.bmp"

4 **Message SHOW** "Question" : "Do you want to see the captured image now?" (other parameters: x = -100, y = -100, Window title = Message, Buttons = Yes and No, Timeout (seconds) = 0, Always on top =).

5 **IF** _vMsgButton==YES

6 **File OPEN** open file "%_vFolder_Personal%\ScreenImage.bmp" in system default viewer.

7 **ENDIF**

Example (Plain Text):

```
<#> This macro shows how to capture display picture
<cmds>

<display_captureimage>(0,0,0,0,"%_vFolder_Personal%\ScreenImage.bmp")

<msg>(-100,-100,"Do you want to see the captured image now?","Message",2,0,1)

<if>("_vMsgButton==YES")
  <fileopen>("%_vFolder_Personal%\ScreenImage.bmp",0)
<endif>
```

NOTIFICATION - < notify >() ... [Pro]

Display NOTIFICATION
<notify>("Message text")
Available in: Professional edition

This command displayes a Windows notification message.

#	Parameter name	Parameter description
1	Message text	The message to display

Example (Macro Steps):

1 <#> <#>This macro shows how to use "notify" command

2 **Macro execution: ONLY COMMANDS**

3 **Date & Time : DATE Insert or save to Variable** Format=Windows user default - long, Separator=/, Day leading zero=Yes, Month leading zero = Yes, Day shift = 0, Variable for result = vDate

4 **Display NOTIFICATION** "Current date: %vDate%"

Example (Plain Text):

```
<#>This macro shows how to use "notify" command
<cmds>
<date>(21,"/",1,1,0,"vDate")
<notify>("Current date: %vDate%")
```

Excel

Read cell value - < excel_cell_get >() ... [Pro]

Excel: Read cell value
<excel_cell_get>(Workbook identifier,Column,Row,Variable for result)
Available in: Professional edition

This command reads a cell value of the currently active worksheet and stores it to variable.

#	Parameter name	Parameter description
1	Workbook identifier	A workbook identifier previously obtained by "open workbook" command.
2	Column	Cell column identifier. For example: F
3	Row	Cell row identifier. For example: 21
4	Variable for result	Variable that receives value from the cell defined by column and row. (Value from F21 cell in our example.)

Example (Macro Steps):

1 <#> <#> This macro shows how to write data to Excel cell

2 <#> <#> and how to read it again

3 **Macro execution: ONLY COMMANDS**

4 **Excel: Open/Create workbook** "" (Worksheet = "", Show = "Yes", Workbook identifier = "wbi")

5 **Excel: Write value to cell** [B 2] <--- B2 value (Workbook identifier = %wbi%)

6 **Excel: Read cell value** [B 2] ---> vCellB2 (Workbook identifier = %wbi%)

7 <#> <#> Let's see what we have written..

8 **Message SHOW** "Information" : "%vCellB2%" (other parameters: x = -100, y = -100, Window title = , Buttons = OK, Timeout (seconds) = 0, Always on top = No).

9 **Excel: Close workbook** "%wbi%" (SAVE = "No")

Example (Plain Text):

```
<#> This macro shows how to write data to Excel cell
<#> and how to read it again
<cmds>
<excel_wb_open>("","",1,wbi)
<excel_cell_set>(%wbi%,B,2,"B2 value")
<excel_cell_get>(%wbi%,B,2,vCellB2)
<#> Let's see what we have written..
<msg>(-100,-100,"%vCellB2%","",1,0,0,0)
<excel_wb_close>(%wbi%,0)
```

Write value to cell - < excel_cell_set >() ... [Pro]

Excel: Write value to cell
<excel_cell_set>(Workbook identifier,Column,Row,"Value")
Available in: Professional edition

This command writes a value to defined cell of currently active worksheet.

#	Parameter name	Parameter description
1	Workbook identifier	A workbook identifier previously obtained by "open workbook" command.
2	Column	Cell column identifier. For example: F
3	Row	Cell row identifier. For example: 21
4	Value	A value to write to the cell.

Example (Macro Steps):

1 <#> <#> This macro shows how to write data to Excel cell

2 <#> <#> and how to read it again

3 **Macro execution: ONLY COMMANDS**

4 **Excel: Open/Create workbook** "" (Worksheet = "", Show = "Yes", Workbook identifier = "wbi")

5 **Excel: Write value to cell** [B 2] <--- B2 value (Workbook identifier = %wbi%)

6 **Excel: Read cell value** [B 2] ---> vCellB2 (Workbook identifier = %wbi%)

7 <#> <#> Let's see what we have written..

8 **Message SHOW** "Information" : "%vCellB2%" (other parameters: x = -100, y = -100, Window title = , Buttons = OK, Timeout (seconds) = 0, Always on top = No).

9 **Excel: Close workbook** "%wbi%" (SAVE = "No")

Example (Plain Text):

```
<#> This macro shows how to write data to Excel cell
<#> and how to read it again
<cmds>
<excel_wb_open>("","",1,wbi)
<excel_cell_set>(%wbi%,B,2,"B2 value")
<excel_cell_get>(%wbi%,B,2,vCellB2)
<#> Let's see what we have written..
<msg>(-100,-100,"%vCellB2%","",1,0,0,0)
<excel_wb_close>(%wbi%,0)
```

Open/Create workbook - < excel_wb_open >() ... [Pro]

Excel: Open/Create workbook
<excel_wb_open>("Workbook","Worksheet",Show,Workbook identifier)
Available in: Professional edition

This command opens a Microsoft ® Excel ® application with defined workbook (file) open (or opens Excel ® with a newly created empty workbook). "Close workbook" command must be used to properly close Excel workbook.
Note: Microsoft ® and Excel ® are Microsoft Corporation registered trademarks.

#	Parameter name	Parameter description
1	Workbook	The workbook (file) to open. If empty then a new workbook is created.
2	Worksheet	The worksheet to activate. If empty then last active worsheet is activated.
3	Show	Excel window is visible: 0 - No 1 - Yes
4	Workbook identifier	Variable that receives identifier of the workbook. This identntifer can be used for "close workbook" and other workbook related commands.

Example (Macro Steps):

1 <#> <#>This macro shows how to open and close an Excel workbook

2 **Macro execution: ONLY COMMANDS**

3 **Excel: Open/Create workbook** "" (Worksheet = "", Show = "Yes", Workbook identifier = "wbi")

4 **Message SHOW** "Information" : "A new Excel workbook is open and will be closed now." (other parameters: x = -100, y = -100, Window title = Excel Open/Close, Buttons = OK, Timeout (seconds) = 0, Always on top = No).

5 **Excel: Close workbook** "%wbi%" (SAVE = "No")

Example (Plain Text):

```
<#>This macro shows how to open and close an Excel workbook
<cmds>
<excel_wb_open>("","",1,wbi)
<msg>(-100,-100,"A new Excel workbook is open and will be closed now.","Excel Open/Close",1,0,0,0)
<excel_wb_close>(%wbi%,0)
```

Save - < excel_wb_save >() ... [Pro]

Excel: Save
<excel_wb_save>(Workbook identifier,"File path")
Available in: Professional edition

This command saves changes made to defined workbook.

#	Parameter name	Parameter description
1	Workbook identifier	A workbook identifier previously obtained by "open workbook" command.
2	File path	File to save the workbook to. If this field is empty then the workbook is saved to the file it was opened from.

Example (Macro Steps):

1 <#> <#>This macro shows how to save Excel workbook

2 **Macro execution: ONLY COMMANDS**

3 **Excel: Open/Create workbook** "" (Worksheet = "", Show = "Yes", Workbook identifier = "wbi")

4 **Excel: Save** Workbook identifier=%wbi%, File path=%_vFolder_Temp%\workbook

5 **Excel: Close workbook** "%wbi%" (SAVE = "No")

6 **Message SHOW** "Information" : "Workbook was saved to this file: %_vFolder_Temp%\workbook" (other parameters: x = -100, y = -100, Window title = Workbook Saved, Buttons = OK, Timeout (seconds) = 0, Always on top = No).

Example (Plain Text):

```
<#>This macro shows how to save Excel workbook
<cmds>
<excel_wb_open>("","",1,wbi)
<excel_wb_save>(%wbi%,"%_vFolder_Temp%\workbook")<excel_wb_close>(%wbi%,0)
<msg>(-100,-100,"Workbook was saved to this file:

%_vFolder_Temp%\workbook","Workbook Saved",1,0,0,0)
```

Get worksheets - < excel_wb_sheets >() ... [Pro]

Excel: Get worksheets
<excel_wb_sheets>(Workbook identifier,Variable for sheet names,Variable for number of sheets,Variable for active sheet)
Available in: Professional edition

This command retrieves names of worksheets (and currently active worksheet) for given workbook.

#	Parameter name	Parameter description
1	Workbook identifier	A workbook identifier previously obtained by "open workbook" command.
2	Variable for sheet names	Variable (array) that receives names of all worksheets.
3	Variable for number of sheets	Variable that receives number of all worksheets.
4	Variable for active sheet	Variable that receives name of currently active worksheet.

Example (Macro Steps):

1 <#> <#>This macro shows how to open and close an Excel workbook

2 **Macro execution: ONLY COMMANDS**

3 **Excel: Open/Create workbook** "" (Worksheet = "", Show = "Yes", Workbook identifier = "wbi")

4 **Excel: Get worksheets** (Workbook identifier = "wbi", Variable for sheet names = "vSheets", Variable for number of sheets = "vSheetNum", Variable for active sheet = "vActiveSheet")

5 **Message SHOW** "Information" : "The newly open workbook has: Number of sheets: %vSheetNum% The first sheet name is: %vSheets[0]% The active sheet name is: %vActiveSheet%" (other parameters: x = -100, y = -100, Window title = Excel Open/Close, Buttons = OK, Timeout (seconds) = 0, Always on top = No).

6 **Excel: Close workbook** "%wbi%" (SAVE = "No")

Example (Plain Text):

```
<#>This macro shows how to open and close an Excel workbook
<cmds>
<excel_wb_open>("","",1,wbi)
<excel_wb_sheets>(wbi,vSheets,vSheetNum,vActiveSheet)
<msg>(-100,-100,"The newly open workbook has:

Number of sheets: %vSheetNum%
The first sheet name is: %vSheets[0]%
The active sheet name is: %vActiveSheet%","Excel Open/Close",1,0,0,0)
<excel_wb_close>(%wbi%,0)
```

Activate worksheet - < excel_wb_activatesheet >() ... [Pro]

Excel: Activate worksheet
<excel_wb_activatesheet>(Workbook identifier,"Worksheet")
Available in: Professional edition

This command activates defined worksheet in an open workbook identified by its identifier.

#	Parameter name	Parameter description
1	Workbook identifier	A workbook identifier previously obtained by "open workbook" command.
2	Worksheet	A name of worksheet to activate.

Example (Macro Steps):

1 <#> <#>This macro shows how to activate worksheet in Excel workbook

2 **Macro execution: ONLY COMMANDS**

3 **Excel: Open/Create workbook** "" (Worksheet = "", Show = "Yes", Workbook identifier = "wbi")

4 **Excel: Activate worksheet** Workbook identifier=%wbi%, Worksheet=Sheet1

5 <#> <#>Do something useful here...

6 **Excel: Close workbook** "%wbi%" (SAVE = "No")

Example (Plain Text):

```
<#>This macro shows how to activate worksheet in Excel workbook
<cmds>
<excel_wb_open>("","",1,wbi)
<excel_wb_activatesheet>(%wbi%,"Sheet1")
<#>Do something useful here...
<excel_wb_close>(%wbi%,0)
```

Close workbook - < excel_wb_close >() ... [Pro]

Excel: Close workbook
<excel_wb_close>(Workbook identifier,SAVE)
Available in: Professional edition

This command closes a Microsoft ® Excel ® workbook (file) and optionaly saves it.
Note: Microsoft ® and Excel ® are Microsoft Corporation registered trademarks.

#	Parameter name	Parameter description
1	Workbook identifier	A workbook identifier previously obtained by "open workbook" command.
2	SAVE	Save workbook: 0 - No 1 - Yes

Example (Macro Steps):

1 <#> <#>This macro shows how to open and close an Excel workbook

2 **Macro execution: ONLY COMMANDS**

3 **Excel: Open/Create workbook** "" (Worksheet = "", Show = "Yes", Workbook identifier = "wbi")

4 **Message SHOW** "Information" : "A new Excel workbook is open and will be closed now." (other parameters: x = -100, y = -100, Window title = Excel Open/Close, Buttons = OK, Timeout (seconds) = 0, Always on top = No).

5 **Excel: Close workbook** "%wbi%" (SAVE = "No")

Example (Plain Text):

```
<#>This macro shows how to open and close an Excel workbook
<cmds>
<excel_wb_open>("","",1,wbi)
<msg>(-100,-100,"A new Excel workbook is open and will be closed now.","Excel Open/Close",1,0,0,0)
<excel_wb_close>(%wbi%,0)
```

External Scripts

Embedded JAVA SCRIPT - < script_js > ... [Pro]

Embedded JAVA SCRIPT
<script_js>
Available in: Professional edition

This command runs a "JavaScript" script within the macro.

Example (Macro Steps):

1	**<#> <#> This is very simple JavaScript example**
2	**Macro execution: ONLY COMMANDS**
3	**Embedded JAVA SCRIPT** \ \ function Hello() \ { \ var WSHShell = WScript.CreateObject("WScript.Shell"); \ WSHShell.Popup("Hello, this is JavaScript"); \ } \ \ Hello() \ \

Example (Plain Text):

```
<#> This is very simple JavaScript example
<cmds>

<script_js>

function Hello()
{
   var WSHShell = WScript.CreateObject("WScript.Shell");
   WSHShell.Popup("Hello, this is JavaScript");
}

Hello()

</script_js>
```

Embedded VB SCRIPT - < script_vbs > ... [Pro]

Embedded VB SCRIPT
<script_vbs>
Available in: Professional edition

This command runs "VBScript" script within the macro.

Example (Macro Steps):

1	**<#> <#> Very simple VB Script example**
2	**Macro execution: ONLY COMMANDS**
3	**Embedded VB SCRIPT** \ MsgBox("This is VB Script") \

Example (Plain Text):

```
<#> Very simple VB Script example
<cmds>
<script_vbs>
MsgBox("This is VB Script")
</script_vbs>
```

Embedded BASIC SCRIPT - < script_basic > ... [Pro]

Embedded BASIC SCRIPT
<script_basic>
Available in: Professional edition

This command runs a "Basic Script" script within the macro. It is possible to exchange data between the program native macro language variables and the Basic Script variables. The variable content exchange is done through DDE - see the samples below.

Example (Macro Steps):

1 <#> <#> Count sin(1), put it to clipboard and show it in Notepad

2 **Macro execution: ONLY COMMANDS**

3 **Embedded BASIC SCRIPT** \ Sub Main \ s = Sin(1) \ Clipboard s \ End Sub \

4 **Run APPLICATION** "notepad.exe" (other parameters: Parameters = , Folder path = , Window state = Normal). Macro execution waits for application to finish up to "0" seconds.

5 **WAIT FOR** Object = "WIN", Event = "OPEN", Parameter = "Notepad", Timeout (seconds) = "5", Exact = "0"

6 **Macro execution: KEYS / FREE TEXT + COMMANDS**

7 **Key** Ctrl

8 **v**

9 **Key** Ctrl

Example (Plain Text):

```
<#> Count sin(1), put it to clipboard and show it in Notepad
<#>
<cmds>
<script_basic>
Sub Main
s = Sin(1)
Clipboard s
End Sub
</script_basic>
<execappex>("notepad.exe","","",0,0)
<waitfor>("WIN","OPEN","Notepad",5,0)
<keys><ctrl>v<ctrl>
```

File Mainpulation

OPEN - < fileopen >() ... [Free]

File OPEN
<fileopen>("File",Window state)
Available in: Free edition

Opens file within the associated application.

#	Parameter name	Parameter description
1	File	Full path to the file to open (e.g., "c:\mydocuments\letter1.doc"). Can be a static text or variable containing text.
2	Window state	The state of the window: 0 - Normal 1 - Maximized 2 - Minimized

Example (Macro Steps):

1 <#> <#> This command opens file user selects

2 **Macro execution: ONLY COMMANDS**

3 **Variable OPERATION** "SELECT_FILE" (Variable for result = vFile, Input text/variable = , Parameter 1 = Select File, Parameter 2 = , Parameter 3 = 0)

4 **IF STRING** _vCanceled==1

5 **Macro EXIT**

6 **ENDIF**

7 **File OPEN** open file "vFile" in system default viewer.

Example (Plain Text):

```
<#> This command opens file user selects
<#>
<cmds>
<var_oper>(vFile,"",SELECT_FILE,"Select File","", "0")
<if_str>("_vCanceled==1") <exitmacro> <endif>
<fileopen>("vFile",0)
```

COPY - < filecopy >() ... [Free]

File COPY
<filecopy>("Source","Destination",Subfolders,Unused,Retries,Variable for number of processed files,Variable for number of failures,"Log errors","Additional options")
Available in: Free edition

This command copies one or multiple files.

#	Parameter name	Parameter description
1	Source	Full path to the file to copy (e.g., "c:\mydocuments\original.doc") or just file name (in such case it is expected the file is located in the same folder as macro file). The file name can contain wildcard characters (*?). In such the case all the files matching the pattern are copied.
2	Destination	Full path to the new file (e.g., "c:\mydocuments\copy.doc") or just file name (in such case the copied file will be located in the same folder as macro file). If the FileSource contains wildcard characters the FileDestination must specify directory (e.g., "c:\mydocuments") where multiple files are copied. This field cannot be left empty.
3	Subfolders	Takes effect only if the FileSource contains wildcard characters. If 1, files from all the sub directories are copied as well (if matching the pattern). If 0, files from sub directories are not copied.
4	Unused	Must be 0.
5	Retries	If the copy fails the command can retry to copy later. This parameter tells the number of retries.
6	Variable for number of processed files	This variable receives the number of files copied. This parameter can be left blank.
7	Variable for number of failures	This variable receives the number of failures. This parameter can be left blank.
8	Log errors	If a file cannot be copied then it is recorded in report file. This parameter can be left blank.
9	Additional options	Additional parameters: "-pr" - show progress window "-on" - overwrite file only if it is newer.

Example (Macro Steps):

```
1   <#> <#> This macro copies file you select to the file "c:\filecopy.txt"
2   Macro execution: ONLY COMMANDS
3   <#> <#> Select file to copy:
4   Variable OPERATION "SELECT_FILE" (Variable for result = vFile, Input text/variable = *.txt, Parameter 1 =
    Select File, Parameter 2 = , Parameter 3 = 0)
5   IF STRING _vCanceled==1
6       Macro EXIT
7   ENDIF
8   <#> <#> Type destination folder:
9   Variable SET "vDestinationFolder=", Message text="Type where to copy the selected file:"
10  IF STRING _vCanceled==1
11      Macro EXIT
12  ENDIF
13  <#> <#> Do not show error messages, we will handle errors programatically
14  Error message DISABLED
15  <#> <#> Copy file
16  File COPY from "vFile" to "%vDestinationFolder%\filecopy.txt" (Subfolders = No, Retries = 3, Variable for
    number of processed files = vCopied, Variable for number of failures = vFailed, Log errors =
    %TEMP%\filecopy_failreport.txt, Additional options = -pr -on )
17  IF vFailed<=0
18      Message SHOW "Information" : "File was copied OK!" (other parameters: x = -100, y = -100, Window title
        = Message, Buttons = OK, Timeout (seconds) = 0, Always on top = ).
19  ELSE activate
20      Message SHOW "Error" : "File copy FAILED!" (other parameters: x = -100, y = -100, Window title =
        Message, Buttons = OK, Timeout (seconds) = 0, Always on top = ).
21  ENDIF
22  Error CLEAR
23  Error message ENABLED
```

Example (Plain Text):

```
<#> This macro copies file you select to the file "c:\filecopy.txt"
<#>
<cmds>

<#> Select file to copy:
```

```
<var_oper>(vFile,"*.txt",SELECT_FILE,"Select File","", "0")
<if_str>("_vCanceled==1")
  <exitmacro>
<endif>

<#> Type destination folder:
<varset>("vDestinationFolder=","Type where to copy the selected file:")
  <if_str>("_vCanceled==1")
<exitmacro>
<endif>


<#> Do not show error messages, we will handle errors programatically
<me_error_nodisplay>

<#> Copy file
<filecopy>("vFile","%vDestinationFolder%\filecopy.txt",0,0,3,vCopied,vFailed,"%TEMP%\filecopy_failreport.txt","-pr -on ")

<if>("vFailed<=0")
  <msg>(-100,-100,"File was copied OK!","Message",1,0,0)
<else>
  <msg>(-100,-100,"File copy FAILED!","Message",1,0,2)
<endif>

<me_error_clear>
<me_error_display>
```

MOVE - < filemove >() ... [Pro]

File MOVE
<filemove>("Source","Destination",Subfolders,Unused,Retries,Variable for number of processed files,Variable for number of failures,"Log errors","Additional options")
Available in: Professional edition

This command moves file(s).

#	Parameter name	Parameter description
1	Source	Full path to the file to move (e.g., "c:\mydocuments\original.doc"). The file path can contain wildcard characters (*?). In such the case all the files matching the pattern are moved. Can be a static text or variable containing text.
2	Destination	Full path to the new file (e.g., "c:\mydocuments\copy.doc"). If the FileSource contains wildcard characters the FileDestination must specify directory (e.g., "c:\mydocuments") where multiple files are moved.
3	Subfolders	Takes effect only if the FileSource contains wildcard characters. If 1, files from all the sub directories are moved as well (if matching the pattern). If 0, files from sub directories are not moved.
4	Unused	Must be 0.
5	Retries	If the move fails the command can retry to move again. This parameter tells the number of retries.
6	Variable for number of processed files	This variable receives the number of files copied. This parameter can be left blank.
7	Variable for number of failures	This variable receives the number of failures. This parameter can be left blank.
8	Log errors	If a file cannot be moved then it is recorded in report file. This parameter can be left blank.
9	Additional options	Additional parameters: "-pr" - show progress window "-on" - overwrite file only if it is newer.

Example (Macro Steps):

```
1   <#> <#> This macro moves file you select to "c:\temp\" folder
2   Macro execution: ONLY COMMANDS
3   Variable OPERATION "SELECT_FILE" (Variable for result = vFile, Input text/variable = *.txt, Parameter 1 = Select File, Parameter 2 = , Parameter 3 = 0)
4   IF STRING _vCanceled==1
5       Macro EXIT
6   ENDIF
7   File MOVE from "vFile" to "c:\temp\" (Subfolders = No, Retries = 0, Variable for number of processed files = , Variable for number of failures = , Log errors = , Additional options = )
```

Example (Plain Text):

```
<#> This macro moves file you select to "c:\temp\"  folder
<#>
<cmds>
<var_oper>(vFile,"*.txt",SELECT_FILE,"Select File","", "0")
<if_str>("_vCanceled==1") <exitmacro> <endif>
<filemove>("vFile","c:\temp\",0,0,0,,,"","")
```

DELETE - < filedel >() ... [Free]

File DELETE
<filedel>("Source",Subfolders,Unused,Retries,Variable for number of processed files,Variable for number of failures,"Log errors",Show progress)
Available in: Free edition

This command deletes specified file(s).

#	Parameter name	Parameter description
1	Source	Full path to the file to delete (e.g., "c:\mydocuments\original.doc"). The file path can contain wildcard characters (*?). In such the case all the files matching the pattern are deleted.
2	Subfolders	Takes effect only if the File contains wildcard characters. If 1, files from all the sub directories are deleted as well (if matching the pattern). If 0, files from sub directories are not deleted.
3	Unused	Must be 0.
4	Retries	
5	Variable for number of processed files	
6	Variable for number of failures	
7	Log errors	
8	Show progress	

Example (Macro Steps):

```
1   <#> <#> This macro deletes the file you select
2   Macro execution: ONLY COMMANDS
3   Variable OPERATION "SELECT_FILE" (Variable for result = vFile, Input text/variable = , Parameter 1 = Select File, Parameter 2 = , Parameter 3 = 0)
4   IF STRING _vCanceled==1
5       Macro EXIT
6   ENDIF
7   File DELETE "vFile" (Subfolders = No, Unused = 0, Retries = , Variable for number of processed files = , Variable for number of failures = , Log errors = , Show progress = )
```

Example (Plain Text):

```
<#> This macro deletes the file you select
<#>
<cmds>
<var_oper>(vFile,"",SELECT_FILE,"Select File","", "0")
<if_str>("_vCanceled==1") <exitmacro> <endif>
<filedel>("vFile",0,0)
```

CREATE - < filecreate >() ... [Pro]

File CREATE
<filecreate>("File",Unused)
Available in: Professional edition

Creates a new empty file.

#	Parameter name	Parameter description
1	File	Full path to the file to create (e.g., "c:\mydocuments\empty.doc"). Can be a static text or variable containing text.
2	Unused	Must be 0.

Example (Macro Steps):

```
1    <#> <#> This example creates new empty file c:\newfile.txt
2    Macro execution: ONLY COMMANDS
3    File CREATE "c:\newfile.txt"
```

Example (Plain Text):

```
<#> This example creates new empty file c:\newfile.txt
<#>
<cmds>
<filecreate>("c:\newfile.txt",0)
```

LOAD TEXT - < data_load >() ... [Pro]

File LOAD TEXT
<data_load>("Variable","File","Password")
Available in: Professional edition

This command loads (textual) data from file (whole the content) to the given variable.

#	Parameter name	Parameter description
1	Variable	Variable that receives the file content (text).
2	File	Path to the file with data (e.g., "c:\mydocuments\data.txt").
3	Password	Password used to decrypt the file content. The password must be the same that was previously used to save data (for example, using 'data_save' or 'data_crypt' command). If incorrect password is provided then the command fails. Leave the parameter empty if the file is not encrypted.

Example (Macro Steps):

1 <#> <#> This macro loads the file content to variable

2 **Macro execution: ONLY COMMANDS**

3 **Variable OPERATION** "SELECT_FILE" (Variable for result = vFile, Input text/variable = , Parameter 1 = Select File, Parameter 2 = , Parameter 3 = 0)

4 **File LOAD TEXT** from file "vFile" to "%vFileData%"

5 **Message SHOW** "" : "vFileData" (other parameters: x = -100, y = -100, Window title = File Content, Buttons = OK, Timeout (seconds) = , Always on top =).

Example (Plain Text):

```
<#> This macro loads the file content to variable
<#>
<cmds>
<var_oper>(vFile,"",SELECT_FILE,"Select File","", "0")
<data_load>("%vFileData%","vFile","")
<msg>(-100,-100,"vFileData","File Content",1)
```

SAVE TEXT - < data_save >() ... [Pro]

File SAVE TEXT
<data_save>("Data","File","Mode","Password")
Available in: Professional edition

Save textual data to file.

#	Parameter name	Parameter description
1	Data	Text to be saved (or variable containing text to be saved).
2	File	Path to the file (e.g., "c:\mydocuments\data.txt").
3	Mode	Can be one of the following values: "A" - the data will be appended to the end of the file "O" - the data will be written to the begin of the file
4	Password	Password to encrypt the file content. Leave the parameter empty if file content encryption is not required.

Example (Macro Steps):

1 <#> <#> This macro adds current date to a new line in the end of the file.

2 **Macro execution: ONLY COMMANDS**

3 **Variable OPERATION** "SELECT_FILE" (Variable for result = vFile, Input text/variable = , Parameter 1 = Select File, Parameter 2 = , Parameter 3 = 0)

4 **Variable SET** "vData=%_vKeyReturn%%_vCurrDate_MMDDYYYY%", Message text=""

5 **File SAVE TEXT** "vData" to file "vFile

Example (Plain Text):

```
<#> This macro adds current date to a new line in the end of the file.
<#>
<cmds>
<var_oper>(vFile,"",SELECT_FILE,"Select File","", "0")
<varset>("vData=%_vKeyReturn%%_vCurrDate_MMDDYYYY%","")
<data_save>("vData","vFile","A")
```

INFO - < fileinfo >() ... [Pro]

File INFO
<fileinfo>("File","Information","Variable for result")
Available in: Professional edition

This command retrieves information (size, creation time, last access time, last modification time) about specified file.

#	Parameter name	Parameter description
1	File	(Full) path to the file (e.g., "c:\mydocuments\file.doc").
2	Information	Type of the information to be retrieved. Can be one of the following values: "SIZE" - retrieves file size "TIME_WRITE_ISO" - retrieves the last modification time in ISO format that can be used for arithmetic and logic operations "TIME_ACCESS_ISO" - retrieves the last access time in ISO format that can be used for arithmetic and logic operations "TIME_CREATE_ISO" - retrieves the creation time in ISO format that can be used for arithmetic and logic operations "TIME_WRITE" - retrieves the last modification time "TIME_ACCESS" - retrieves the last access time "TIME_CREATE" - retrieves the creation time "FILE_NAME" - retrieves the name of the file "FILE_NAME_NO_EXTENSION" - retrieves the name of the file without extension "FILE_EXTENSION" - retrieves the file extension "FILE_PATH" - retrieves the file full path
3	Variable for result	Variable that receives the result.

Example (Macro Steps):

1 <#> <#> This macro retrieves information about the file you select

2 **Macro execution: ONLY COMMANDS**

3 **Variable OPERATION** "SELECT_FILE" (Variable for result = vFile, Input text/variable = , Parameter 1 = Select File, Parameter 2 = , Parameter 3 = 0)

4 **IF FILE** "%vFile%" Exist (0)

5 **File INFO** : save "Size" information of file "vFile" to variable "vFileSize"

6 **File INFO** : save "Last modified time" information of file "vFile" to variable "vFileTime"

7 **Message SHOW** "" : "File size is: %vFileSize% Last modification time is: %vFileTime%" (other parameters: x = -100, y = -100, Window title = File Info, Buttons = OK, Timeout (seconds) = , Always on top =).

8 **ENDIF**

Example (Plain Text):

```
<#> This macro retrieves information about the file you select
<cmds>
<var_oper>(vFile,"",SELECT_FILE,"Select File","", "0")
<if_file>("%vFile%","EXIST","0")<#>
  <fileinfo>("vFile","SIZE","vFileSize")
```

```
   <fileinfo>("vFile","TIME_WRITE","vFileTime")
   <msg>(-100,-100,"File size is: %vFileSize%
Last modification time is: %vFileTime%","File Info",1)<#>
<endif>
```

ENUMERATE - < file_enum >() ... [Pro]

File ENUMERATE
<file_enum>("Folder",Option,Variable array for enumerated items,Variable array size)
Available in: Professional edition

This command enumerates files/folders from the given folder using a defined mask (wildcards).

#	Parameter name	Parameter description
1	Folder	File/folder selection pattern containing wildcards. Example: "c:\windows\a*.?if" or "c:\windows\sys*".
2	Option	1 - only subfolders are enumerated 2 - only files are enumerated 3 - both subfolders and files are enumerated
3	Variable array for enumerated items	Variable (array) that receives files/folders from the Folder that match the mask.
4	Variable array size	Variable that receives the number of files/folders saved in the VarFiles.

Example (Macro Steps):

1 <#> <#> This macro shows how to use 'file_enum' command.

2 **Macro execution: ONLY COMMANDS**

3 **File ENUMERATE** "Files only" of "Folder=%_vFolder_Temp%*.tmp" (Variable array for enumerated items=vFiles, Variable array size=vNumOfFiles)

4 **Message SHOW** "" : "There are %vNumOfFiles% TMP files in "%_vFolder_Temp%" directory." (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

5 **Variable SET** "vMsg=_vStrEmpty", Message text=""

6 **Loop BEGIN** Repeat = vNumOfFiles

7 **Variable OPERATION** "STR_APPEND" (Variable for result = vMsg, Input text/variable = %vMsg%, Parameter 1 = %_vKeyReturn%%vFiles[_vLoopCounter0]%, Parameter 2 = , Parameter 3 = 0)

8 **Loop END**

9 **Message SHOW** "" : "vMsg" (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

Example (Plain Text):

```
<#> This macro shows how to use 'file_enum' command.
<#>
<cmds>
<file_enum>("%_vFolder_Temp%\*.tmp",2,vFiles,vNumOfFiles)
<msg>(-100,-100,"There are %vNumOfFiles% TMP files in %_vQuoteChar%%_vFolder_Temp%%_vQuoteChar%
directory.","Message",1)
<varset>("vMsg=_vStrEmpty","")
```

```
<begloop>(vNumOfFiles)
   <var_oper>(vMsg,"%vMsg%",STR_APPEND,"%_vKeyReturn%%vFiles[_vLoopCounter0]%","", "0")
<endloop>

<msg>(-100,-100,"vMsg","Message",1)
```

PRINT - < file_print >() ... [Pro]

File PRINT
<file_print>(File)
Available in: Professional edition

Prints specified file on a default printer.
Note: There must be a "Print" command associated with the document type in Windows.

#	Parameter name	Parameter description
1	File	Full path to the file to be printed.

Example (Macro Steps):

```
1    <#> <#> This macro prints a file
2    Macro execution: ONLY COMMANDS
3    File PRINT "c:\MyDocuments\table.xlsx"
```

Example (Plain Text):

```
<#> This macro prints a file
<#>
<cmds>
<file_print>(c:\MyDocuments\table.xlsx)
```

RENAME - < filerename >() ... [Pro]

File RENAME
<filerename>("File","New name",Unused,Unused)
Available in: Professional edition

This command renames specified file.

#	Parameter name	Parameter description
1	File	(Full) path to the file to copy (e.g., "c:\mydocuments\original.doc").
2	New name	New name such as "backup.doc".
3	Unused	Must be 0.
4	Unused	Must be 0.

Example (Macro Steps):

```
1    <#> <#> This macro renames file you select to "_renamed.txt".
2    Macro execution: ONLY COMMANDS
3    Variable OPERATION "SELECT_FILE" (Variable for result = vFile, Input text/variable = , Parameter 1 = Select File, Parameter 2 = , Parameter 3 = 0)
4    IF STRING _vCanceled==1
5        Macro EXIT
6    ENDIF
7    File RENAME vFile, New name = _renamed.txt,Unused = 0
```

Example (Plain Text):

```
<#> This macro renames file you select to "_renamed.txt".
<#>
<cmds>
<var_oper>(vFile,"",SELECT_FILE,"Select File","", "0")
<if_str>("_vCanceled==1") <exitmacro> <endif>
<filerename>("vFile","_renamed.txt",0,0)
```

ZIP - < zip_createfile >() ... [Pro]

File ZIP
<zip_createfile>("Zip file","Files to archive",Subfolders,"Password")
Available in: Professional edition

This command archives one or multiple files to a single ZIP file.

#	Parameter name	Parameter description
1	Zip file	(Full) path to the ZIP file to be created. Example: c:\temp\zipfile.zip.
2	Files to archive	(Full) file path or wildcard mask of files that will be included (zipped) to the ZIP file. Example: c:\temp*.txt.
3	Subfolders	If 1, also files from subfolders that match the "Input files mask" are included. If 0, the files from subfolders are not included.
4	Password	If a password is provided then the zip file created is password protected.

Example (Macro Steps):

```
1   <#> <#> This example shows how to zip multiple files to a single ZIP file.
2   <#> <#>
3   Macro execution: ONLY COMMANDS
4   <#> <#> Create folder for sample files
5   Folder CREATE "c:\temp\zip_sample"
6   <#> <#> Create sample files
7   File CREATE "c:\temp\zip_sample\file1.txt"
8   File CREATE "c:\temp\zip_sample\file2.txt"
9   File CREATE "c:\temp\zip_sample\file3.txt"
10  <#> <#> Zip files to a zip file
11  File ZIP "c:\temp\zip_sample\*.txt" to "c:\temp\zip_sample\zipfile.zip", Subfolders = No, Password =
12  <#> <#> Open folder to see the results
13  Folder OPEN open folder "c:\temp\zip_sample" in Windows Explorer.
```

Example (Plain Text):

```
<#> This example shows how to zip multiple files to a single ZIP file.
<#>
<cmds>

<#> Create folder for sample files
<dircreate>("c:\temp\zip_sample",0)
```

```
<#> Create sample files
<filecreate>("c:\temp\zip_sample\file1.txt",0)
<filecreate>("c:\temp\zip_sample\file2.txt",0)
<filecreate>("c:\temp\zip_sample\file3.txt",0)

<#> Zip files to a zip file
<zip_createfile>("c:\temp\zip_sample\zipfile.zip","c:\temp\zip_sample\*.txt",0, "")

<#> Open folder to see the results
<diropen>("c:\temp\zip_sample",0)
```

UNZIP - < zip_unzipfile >() ... [Pro]

File UNZIP
<zip_unzipfile>("Zip file","Destination","Mask files","Password")
Available in: Professional edition

This command extracts from ZIP file to a folder the files that match defined mask.

#	Parameter name	Parameter description
1	Zip file	(Full) path to an existing ZIP file. Example: c:\temp\zipfile.zip.
2	Destination	(Full) path to a folder where the files contained in the ZIP file will be extracted. If the folder does not exist then it is created automatically. Example: c:\temp\unzipDir.
3	Mask files	Wildcard mask to define what files will be extracted. Only mask matching files are extracted. Example: *.txt.
4	Password	If the zip file is password protected then the password parameter is needed to successfully unzip the file.

Example (Macro Steps):

```
1   <#> <#> This example shows how to unzip existing ZIP file.
2   <#> <#>
3   Macro execution: ONLY COMMANDS
4   <#> <#> Create folder for sample files
5   Folder CREATE "c:\temp\zip_sample"
6   <#> <#> Create sample files
7   File CREATE "c:\temp\zip_sample\file1.txt"
8   File CREATE "c:\temp\zip_sample\file2.txt"
9   File CREATE "c:\temp\zip_sample\file3.txt"
10  <#> <#> Zip files to a zip file
11  File ZIP "c:\temp\zip_sample\*.txt" to "c:\temp\zip_sample\zipfile.zip", Subfolders = No, Password =
12  <#> <#> Unzip file to folder
13  File UNZIP to "c:\temp\zip_sample\unzip" from "c:\temp\zip_sample\zipfile.zip", Mask files = *.txt, Password
    =
14  <#> <#> Open folder to see the results
15  Folder OPEN open folder "c:\temp\zip_sample\unzip" in Windows Explorer.
```

Example (Plain Text):

<#> This example shows how to unzip existing ZIP file.
<#>
<cmds>

<#> Create folder for sample files
<dircreate>("c:\temp\zip_sample",0)

<#> Create sample files
<filecreate>("c:\temp\zip_sample\file1.txt",0)
<filecreate>("c:\temp\zip_sample\file2.txt",0)
<filecreate>("c:\temp\zip_sample\file3.txt",0)

<#> Zip files to a zip file
<zip_createfile>("c:\temp\zip_sample\zipfile.zip","c:\temp\zip_sample*.txt",0)

<#> Unzip file to folder
<zip_unzipfile>("c:\temp\zip_sample\zipfile.zip","c:\temp\zip_sample\unzip","*.txt")

<#> Open folder to see the results
<diropen>("c:\temp\zip_sample\unzip",0)

CREATE SELF-EXTRACTING ZIP - < zip_create_sfx >() ... [Pro]

File CREATE SELF-EXTRACTING ZIP
<zip_create_sfx>("Self-extracting file","Files to archive",Subfolders,"Startup file",Option,"Password")
Available in: Professional edition

This command archives one or multiple files to a single self-extracting ZIP file.

#	Parameter name	Parameter description
1	Self-extracting file	(Full) path to the self-extracting ZIP file to be created. Example: c:\temp\zipfile-se.exe.
2	Files to archive	(Full) file path or wildcard mask of files that will be included (zipped) to the ZIP file. Example: c:\temp*.txt.
3	Subfolders	If 1, also files from subfolders that match the "Input files mask" are included. If 0, the files from subfolders are not included.
4	Startup file	Full path of the file to automatically open when the self-extracting ZIP file finishes files extraction. Example: c:\temp\readme.txt.
5	Option	If 1, the files are extracted automatically to the default temporary folder. If 0, the user will select the folder where to extract the files.
6	Password	If a password is provided then the exe file created is password protected.

Example (Macro Steps):

```
1   <#> <#> This example shows how to create self-extracting ZIP file.
2   <#> <#>
3   Macro execution: ONLY COMMANDS
4   <#> <#> Create folder for sample files
5   Folder CREATE "c:\temp\zip_sample"
6   <#> <#> Create sample files
7   File CREATE "c:\temp\zip_sample\file1.txt"
8   File CREATE "c:\temp\zip_sample\file2.txt"
9   File CREATE "c:\temp\zip_sample\file3.txt"
10  <#> <#> Create self-extracting ZIP file
11  File CREATE SELF-EXTRACTING ZIP "c:\temp\zip_sample\zipfile-selfextract.exe" from
    "c:\temp\zip_sample\*.txt", Subfolders = No, Startup file = c:\temp\zip_sample\file1.txt, Option = User will
    select destination folder, Password =
12  <#> <#> Open folder to see the results
13  Folder OPEN open folder "c:\temp\zip_sample" in Windows Explorer.
```

Example (Plain Text):

```
<#> This example shows how to create self-extracting ZIP file.
<#>
<cmds>

<#> Create folder for sample files
<dircreate>("c:\temp\zip_sample",0)

<#> Create sample files
<filecreate>("c:\temp\zip_sample\file1.txt",0)
<filecreate>("c:\temp\zip_sample\file2.txt",0)
<filecreate>("c:\temp\zip_sample\file3.txt",0)

<#> Create self-extracting ZIP file
<zip_create_sfx>("c:\temp\zip_sample\zipfile-selfextract.exe","c:\temp\zip_sample\*.txt",0,"c:\temp\zip_sample\file1.txt",0)

<#> Open folder to see the results
<diropen>("c:\temp\zip_sample",0)
```

.INI WRITE - < ini_file_write >() ... [Pro]

File .INI WRITE
<ini_file_write>("Section","Key","Data","File","Password")
Available in: Professional edition

The command writes data to a INI file. INI files internal structure look like this:
...

[Section]
Key1=Some data.
Key2=Other data.
...
KeyN=Even more data.

[Other Section]
Key1=....

...

The "ini_file_write" command writes data to given key in given section.

#	Parameter name	Parameter description
1	Section	Section name in INI file.
2	Key	Name of the key in the Section.
3	Data	Data to be written to the given key in given section.
4	File	(Full) path to the INI file.
5	Password	Password to encrypt the data. Leave the parameter empty if data encryption is not required.

Example (Macro Steps):

```
1   <#> <#> This sample writes and reads data from INI file.
2   <#> <#>
3   Macro execution: ONLY COMMANDS
4   <#> <#> Write some data to the INI file first
5   File .INI WRITE Section = "section1", Key = "key1", Data = "Value1", File = "%TEMP%\example.ini",
    Password = ""
6   <#> <#> Read the data then...
7   File .INI READ Section = "section1", Key = "key1", Variable to save data = "Value", File =
    "%TEMP%\example.ini", Password = ""
8   <#> <#> ... and show the data.
9   Message SHOW "Information" : "Value" (other parameters: x = -100, y = -100, Window title = Message,
    Buttons = OK, Timeout (seconds) = 0, Always on top = ).
10  <#> <#> In the end let's delete the ini file.
11  File DELETE "%TEMP%\example.ini" (Subfolders = No, Unused = 0, Retries = 3, Variable for number of
    processed files = , Variable for number of failures = , Log errors = , Show progress = No)
```

Example (Plain Text):

```
<#> This sample writes and reads data from INI file.
<#>
<cmds>

<#> Write some data to the INI file first
<ini_file_write>("section1","key1","Value1","%TEMP%\example.ini")

<#> Read the data then...
<ini_file_read>("section1","key1","Value","%TEMP%\example.ini")

<#> ... and show the data.
<msg>(-100,-100,"Value","Message",1,0,0)

<#> In the end let's delete the ini file.
<filedel>("%TEMP%\example.ini",0,0,3,,,"",0)
```

.INI READ - < ini_file_read >() ... [Pro]

File .INI READ
<ini_file_read>("Section","Key","Variable to save data","File","Password")
Available in: Professional edition

The command reads data from a INI file. INI files internal structure look like this:
...

[Section]
Key1=Some data.
Key2=Other data.
...
KeyN=Even more data.

[Other Section]
Key1=....

...

The "ini_file_read" command reads data from given key in given section.

#	Parameter name	Parameter description
1	Section	Section name in INI file.
2	Key	Name of the key in the Section.
3	Variable to save data	Variable that receives data retrieved from the given key in given section.
4	File	(Full) path to the INI file.
5	Password	Password used to decrypt the data. The password must be the same that was previously used to encrypt data (in 'ini_file_write' or 'data_crypt' command). If incorrect password is provided then the command fails. Leave the parameter empty if the data are not encrypted.

Example (Macro Steps):

```
1   <#> <#> This sample writes and reads data from INI file.
2   <#> <#>
3   Macro execution: ONLY COMMANDS
4   <#> <#> Write some data to the INI file first
5   File .INI WRITE Section = "section1", Key = "key1", Data = "Value1", File = "%TEMP%\example.ini",
    Password = ""
6   <#> <#> Read the data then...
7   File .INI READ Section = "section1", Key = "key1", Variable to save data = "Value", File =
    "%TEMP%\example.ini", Password = ""
8   <#> <#> ... and show the data.
9   Message SHOW "Information" : "Value" (other parameters: x = -100, y = -100, Window title = Message,
    Buttons = OK, Timeout (seconds) = 0, Always on top = ).
10  <#> <#> In the end let's delete the ini file.
11  File DELETE "%TEMP%\example.ini" (Subfolders = No, Unused = 0, Retries = 3, Variable for number of
    processed files = , Variable for number of failures = , Log errors = , Show progress = No)
```

Example (Plain Text):

```
<#> This sample writes and reads data from INI file.
<#>
<cmds>

<#> Write some data to the INI file first
<ini_file_write>("section1","key1","Value1","%TEMP%\example.ini")

<#> Read the data then...
<ini_file_read>("section1","key1","Value","%TEMP%\example.ini")

<#> ... and show the data.
<msg>(-100,-100,"Value","Message",1,0,0)

<#> In the end let's delete the ini file.
<filedel>("%TEMP%\example.ini",0,0,3,,,"",0)
```

ENCRYPT/DECRYPT - < file_encryption >() ... [Pro]

File ENCRYPT/DECRYPT

<file_encryption>("Input","Output",Information,Encryption bit strength,"Password")
Available in: Professional edition

This command encrypts/decrypts a single file.

#	Parameter name	Parameter description
1	Input	(Full) path to a file that is to be encrypted/decrypted. Example: c:\temp\myPrivateInfo.txt.
2	Output	(Full) path to a file that is created as a result of the encryption/decryption of the "Input File". Example: c:\temp\myPrivateInfo.txt.aes.
3	Information	This parameter can be one of these: ENCRYPT_AES - the input file is encrypted using AES. DECRYPT_AES - the input file (previously encrypted using ENCRYPT_AES) is decrypted.
4	Encryption bit strength	Strength of the encryption. These values are supported: 128, 192, 256.
5	Password	User defined password that is used to encrypt/decrypt the input file. A file encrypted by a password can be successfully decrypted again only if the same password is used.

Example (Macro Steps):

```
1   <#> <#> This simple example shows how to encrypt/decrypt file.
2   <#> <#>
3   Macro execution: ONLY COMMANDS
4   <#> <#> Create folder for sample files
5   Folder CREATE "c:\temp\file_encryption_sample"
6   <#> <#> Create sample file
7   File SAVE TEXT "This is a sample file." to file "c:\temp\file_encryption_sample\1.file_original.txt
8   <#> <#> Now encrypt the file...
9   File ENCRYPT/DECRYPT Input = "c:\temp\file_encryption_sample\1.file_original.txt", Output =
    "c:\temp\file_encryption_sample\2.file_encrypted.txt", Information = "ENCRYPT_AES", Encryption bit strength
    = "128"
10  <#> <#> ...and decrypt again
11  File ENCRYPT/DECRYPT Input = "c:\temp\file_encryption_sample\2.file_encrypted.txt", Output =
    "c:\temp\file_encryption_sample\3.file_decrypted.txt", Information = "DECRYPT_AES", Encryption bit strength
    = "128"
12  <#> <#> Open folder to see the results
13  Folder OPEN open folder "c:\temp\file_encryption_sample" in Windows Explorer.
```

Example (Plain Text):

```
<#> This simple example shows how to encrypt/decrypt file.
<#>
<cmds>

<#> Create folder for sample files
<dircreate>("c:\temp\file_encryption_sample",0)

<#> Create sample file
<data_save>("This is a sample file.","c:\temp\file_encryption_sample\1.file_original.txt","")

<#> Now encrypt the file...
<file_encryption>("c:\temp\file_encryption_sample\1.file_original.txt","c:\temp\file_encryption_sample\2.file_encrypted.txt",E
NCRYPT_AES,128,"nyrangers")

<#> ...and decrypt again
<file_encryption>("c:\temp\file_encryption_sample\2.file_encrypted.txt","c:\temp\file_encryption_sample\3.file_decrypted.txt"
,DECRYPT_AES,128,"nyrangers")

<#> Open folder to see the results
<diropen>("c:\temp\file_encryption_sample",0)
```

Parse Path - < file_path_parse >() ... [Pro]

File Parse Path
<file_path_parse>("File path",Variable for Drive or Server,Variable for Folder,Variable for Name,Variable for Extension)
Available in: Professional edition

This command parses (full qualified) file path to these parts: drive or network server, folder, file name, and file extension.

#	Parameter name	Parameter description
1	File path	File path to be parsed. For example: c:\my folder\sub1\sub2\results.doc
2	Variable for Drive or Server	Variable that receives drive or network server address part of the file path. For example: c
3	Variable for Folder	Variable that receives folder part of the file path. For example: my folder\sub1\sub2
4	Variable for Name	Variable that receives file name part of the file path. For example: results
5	Variable for Extension	Variable that receives extension part of the file path. For example: doc

Example (Macro Steps):

1 <#> <#>This macro breaks 'c:\program files\any program\application.exe' file path to parts

2 **Macro execution: ONLY COMMANDS**

3 **File Parse Path** File path = "c:\program files\any program\application.exe", Variable for Drive or Server = "vDrive", Variable for Folder = "vFolder", Variable for Name = "vName", Variable for Extension = "vExt"

4 **Message SHOW** "Information" : "Drive: %vDrive% Folder: %vFolder% File: %vName% Extension: %vExt%" (other parameters: x = -100, y = -100, Window title = , Buttons = OK, Timeout (seconds) = 0, Always on top = No).

Example (Plain Text):

```
<#>This macro breaks 'c:\program files\any program\application.exe' file path to parts
<cmds>
<file_path_parse>("c:\program files\any program\application.exe",vDrive,vFolder,vName,vExt)
<msg>(-100,-100,"Drive: %vDrive%
Folder: %vFolder%
File: %vName%
Extension: %vExt%","",1,0,0,0)
```

Convert HTML to XML - < file_html2xml >() ... [Pro]

File Convert HTML to XML
<file_html2xml>("HTML File","XML File")
Available in: Professional edition

This command converts HTML file to XML file. HTML files are not always XML syntax compliant. This command makes an HTML file XML compliant so that XML maco commands can be used to parse the file content.

#	Parameter name	Parameter description
1	HTML File	(Full) path to the input HTML file.
2	XML File	(Full) path to the output XML file.

Example (Macro Steps):

1 <#> <#> This example shows how to use HTML to XML command.

2 **Macro execution: ONLY COMMANDS**

3 **Http DOWNLOAD** "http://www.macrotoolworks.com" (Login name=) to file "macrotoolworks.html"

4 **File Convert HTML to XML** Convert HTML file macrotoolworks.html to XML file macrotoolworks.xml

5 **XML File Open** File path = "macrotoolworks.xml", File handle variable = "vXmlDoc", Root element handle variable = "vXmlDocRoot", Password = ""

6 **XML Navigate to Element** (Element handle variable = "%vXmlDoc%", Path = "root\html\body\div\div\div\div\h1\a", Element handle variable = "vTestElem")

7 **XML Attribute Get** (Element handle variable = "%vTestElem%", Attribute name = "title", Variable for result = "vHtmlPageTitle")

8 **Message SHOW** "Information" : "%vHtmlPageTitle%" (other parameters: x = -100, y = -100, Window title = , Buttons = OK, Timeout (seconds) = 0, Always on top = No).

Example (Plain Text):

```
<#> This example shows how to use HTML to XML command.
<#>
<cmds>
<download>("macrotoolworks.html","http://www.macrotoolworks.com","","")
<file_html2xml>("macrotoolworks.html","macrotoolworks.xml")
<xml_file_open>("macrotoolworks.xml",vXmlDoc,vXmlDocRoot)
<xml_element_navigate>(%vXmlDoc%,"root\html\body\div\div\div\div\h1\a",vTestElem)
<xml_attribute_get>(%vTestElem%,title,vHtmlPageTitle)
<msg>(-100,-100,"%vHtmlPageTitle%","",1,0,0,0)
```

CSV Load - < csv_file_load >() ... [Pro]

File CSV Load
<csv_file_load>("File path",File handle variable)
Available in: Professional edition

This command loads CSV file data. When data are loaded then "get record" command can be used in a loop to get fields for each record.

#	Parameter name	Parameter description
1	File path	(Full) path to the input CSV file.
2	File handle variable	Variable that receives CSV file handle. This variable is used as input parameter to "get record" command.

Example (Macro Steps):

```
1   <#> <#> This example shows how to parse CSV file
2   Macro execution: ONLY COMMANDS
3   <#> <#>Load CSV file
4   File CSV Load c:\data\file1.csv, File handle variable = vCsvFile,%p3_name =
5   <#> <#>Read all records in loop
6   Repeat steps UNTIL "1" (Counter variable initial value = "i=0", Counter loop increment = "1")
7       File CSV Get Record Fields (File handle variable = "vCsvFile", Variable array for fields = "vFields",
        Variable for number of fields = "vFieldsNum")
8       IF %vFieldsNum%<=0
9           <#> <#>No more records to process
10          Repeat steps BREAK
11      ENDIF
12      Repeat steps UNTIL "%r%
```

CSV Get Record Fields - < csv_get_record >() ... [Pro]

File CSV Get Record Fields
<csv_get_record>(File handle variable,Variable array for fields,Variable for number of fields)
Available in: Professional edition

This command retrieves one record from CSV file. Call the command multiple times in order to read all records.

Note: CSV (comma separated value) file is a textual file consisting of multiple rows (records) where on each row there are multiple entries (fields) separated by comma:
Read more here: https://en.wikipedia.org/wiki/Comma-separated_values

#	Parameter name	Parameter description
1	File handle variable	Variable containing CSV file handle obtained by "CSV load data" command.
2	Variable array for fields	Variable array that receives fields for the current record.
3	Variable for number of fields	Variable that receives number of fields obtained. If this value is 0 then no more data is in the CSV file.

Example (Macro Steps):

```
1   <#> <#> This example shows how to parse CSV file
2   Macro execution: ONLY COMMANDS
3   <#> <#>Load CSV file
4   File CSV Load c:\data\file1.csv, File handle variable = vCsvFile,%p3_name =
5   <#> <#>Read all records in loop
6   Repeat steps UNTIL "1" (Counter variable initial value = "i=0", Counter loop increment = "1")
7       File CSV Get Record Fields (File handle variable = "vCsvFile", Variable array for fields = "vFields",
        Variable for number of fields = "vFieldsNum")
8       IF %vFieldsNum%<=0
9           <#> <#>No more records to process
10          Repeat steps BREAK
11      ENDIF
12      Repeat steps UNTIL "%r%
```

SHORTCUT - < file_shortcut >() ... [Pro]

File SHORTCUT
<file_shortcut>("Shortcut file","File path","Parameters","Description","Icon file","Icon index")
Available in: Professional edition

This command creates a shortcut file (link - .lnk file) to a file or folder.

#	Parameter name	Parameter description
1	Shortcut file	Full path to the newly created shortcut file (e.g., "c:\mydocuments\shortcut.lnk") or just a shortcut file name - in such case the shortcut file will be created in the same folder where macro file is located).
2	File path	Full path to the file or folder the shortcut is created for. If just file or folder name is provided then it is expected the file or folder is located in the same folder where macro file is located).
3	Parameters	Parameters passed to the file when the shortcut is open. This typically applies to executables.
4	Description	Description of the shortcut.
5	Icon file	Full path to the icon file. If just the icon file name is provided then it is expected the file is located in the same folder where macro file is located).
6	Icon index	Index of the icon within the icon file (if the file contains multiple icons).

Example (Macro Steps):

1 <#> <#> This example shows how to create a shortcut file (.lnk) to other file or folder

2 **File SHORTCUT** "ShortcutToNotesDoc.lnk" to "c:\my documents\notes.doc" (Parameters=)

Example (Plain Text):

<#> This example shows how to create a shortcut file (.lnk) to other file or folder
<file_shortcut>("ShortcutToNotesDoc.lnk","c:\my documents\notes.doc","","Notes document","","0")

Folder Manipulation

OPEN - < diropen >() ... [Pro]

Folder OPEN
<diropen>("Folder",Window state)
Available in: Professional edition

Opens directory in Windows Explorer.

#	Parameter name	Parameter description
1	Folder	Full path to the directory to be opened (e.g., "c:\mydocuments").
2	Window state	The state of the window: 0 - Normal 1 - Maximized 2 - Minimized

Example (Macro Steps):

1 <#> <#> This macro opens directory "c:\"

2 **Macro execution: ONLY COMMANDS**

3 **Folder OPEN** open folder "c:\" in Windows Explorer.

Example (Plain Text):

```
<#> This macro opens directory "c:\"
<#>
<cmds>
<diropen>("c:\",1)
```

CREATE - < dircreate >() ... [Free]

Folder CREATE
<dircreate>("Folder",Unused)
Available in: Free edition

Creates new directory.

#	Parameter name	Parameter description
1	Folder	Full path to the directory to be created (e.g., "c:\mydocuments\new").
2	Unused	Must be 0.

Example (Macro Steps):

1 <#> <#> This macro creates new directory "c:\newdir\temp"

2 **Macro execution: ONLY COMMANDS**

3 **Folder CREATE** "c:\newdir\temp"

Example (Plain Text):

```
<#> This macro creates new directory "c:\newdir\temp"
<#>
<cmds>
<dircreate>("c:\newdir\temp",0)
```

DELETE - < dirdel >() ... [Free]

Folder DELETE
<dirdel>("Source",Unused,Retries,Variable for number of processed files,Variable for number of failures,"Log errors",Show progress)
Available in: Free edition

Deletes directory including all subdirectories. The directory doesn't have to be empty - all the files in the directory are deleted.

#	Parameter name	Parameter description
1	Source	Full path to the directory to delete (e.g., "c:\olddata").
2	Unused	Must be 0.
3	Retries	
4	Variable for number of processed files	
5	Variable for number of failures	
6	Log errors	
7	Show progress	

Example (Macro Steps):

```
1    <#> <#> This macro deletes directory you select
2    Macro execution: ONLY COMMANDS
3    Message SHOW "" : "Select folder you want to delete." (other parameters: x = 100, y = 100, Window title = Message, Buttons = None, Timeout (seconds) = , Always on top = ).
4    Variable OPERATION "SELECT_FOLDER" (Variable for result = vDir, Input text/variable = , Parameter 1 = Select Folder, Parameter 2 = , Parameter 3 = 0)
5    IF STRING _vCanceled==1
6        Macro EXIT
7    ENDIF
8    Message CLOSE
9    Message SHOW "" : "Folder delete process is in progress. Please wait...." (other parameters: x = 100, y = 100, Window title = Message, Buttons = None, Timeout (seconds) = , Always on top = ).
10   Folder DELETE "vDir" (Unused = 0, Variable for number of processed files = , Variable for number of failures = , Log errors = , Show progress = )
11   Message CLOSE
```

Example (Plain Text):

```
<#> This macro deletes directory you select
<#>
```

```
<cmds>

<msg>(100,100,"Select folder you want to delete.","Message",0)
<var_oper>(vDir,"",SELECT_FOLDER,"Select Folder","", "0")
<if_str>("_vCanceled==1")
   <exitmacro>
<endif>
<msgoff>

<msg>(100,100,"Folder delete process is in progress. Please wait....","Message",0)
<dirdel>("vDir",0)
<msgoff>
```

COPY - < dircopy >() ... [Pro]

Folder COPY

<dircopy>("Source","Destination",Subfolders,Unused,Retries,Variable for number of processed files,Variable for number of failures,"Log errors","Additional options")
Available in: Professional edition

Copies directory including all subdirectories. The command continues copying files even if some files fails to be copied. Failed files are reported in report file and also number of failures can be saved in user defined variable so that errors can be handled programmatically.

#	Parameter name	Parameter description
1	Source	Full path to the source directory (e.g., "c:\mydocuments").
2	Destination	Full path to the destination directory (e.g., "c:\backup").
3	Subfolders	Must be 0.
4	Unused	Must be 0.
5	Retries	If the copy fails the command can retry to copy later. This parameter tells the number of retries.
6	Variable for number of processed files	This variable receives the number of files copied. This parameter can be left blank.
7	Variable for number of failures	This variable receives the number of failures. This parameter can be left blank.
8	Log errors	If a file cannot be copied then it is recorded in report file. This parameter can be left blank.
9	Additional options	Additional parameters: "-pr" - show progress window "-on" - overwrite file only if it is newer.

Example (Macro Steps):

```
1   <#> <#> This macro copies directory you select to other directory you select.
2   Macro execution: ONLY COMMANDS
3   Message SHOW "" : "Select folder you want to copy." (other parameters: x = 100, y = 100, Window title =
    Message, Buttons = None, Timeout (seconds) = , Always on top = ).
4   Variable OPERATION "SELECT_FOLDER" (Variable for result = vDirSource, Input text/variable = , Parameter
    1 = Select Source Folder, Parameter 2 = , Parameter 3 = 0)
5   Procedure CALL: ExitOnCancel with parameters ()
6   Message CLOSE
7   Message SHOW "" : "Select destination folder." (other parameters: x = 100, y = 100, Window title =
    Message, Buttons = None, Timeout (seconds) = , Always on top = ).
8   Variable OPERATION "SELECT_FOLDER" (Variable for result = vDirDest, Input text/variable = , Parameter 1
    = Select Destination Folder, Parameter 2 = , Parameter 3 = 0)
9   Procedure CALL: ExitOnCancel with parameters ()
10  Message CLOSE
11  Message SHOW "" : "Folder copy is in progress. Please wait...." (other parameters: x = 100, y = 100,
    Window title = Message, Buttons = None, Timeout (seconds) = , Always on top = ).
12  Folder COPY from "vDirSource" to "vDirDest" (Subfolders = No, Retries = 3, Variable for number of processed
    files = , Variable for number of failures = vFail, Log errors = %TEMP%\DirCopyFailuresReport.txt, Additional
    options = -pr -on )
13  Message CLOSE
14  <#> <#> ------------------------------------------------------------
15  Procedure BEGIN: ExitOnCancel with parameters ()
16      IF STRING _vCanceled==1
17          Macro EXIT
18      ENDIF
19  Procedure END
20  <#> <#> ------------------------------------------------------------
```

Example (Plain Text):

```
<#> This macro copies directory you select to other directory you select.
<#>
<cmds>

<msg>(100,100,"Select folder you want to copy.","Message",0)
<var_oper>(vDirSource,"",SELECT_FOLDER,"Select Source Folder","", "0")
<proc_call>(ExitOnCancel,)
<msgoff>

<msg>(100,100,"Select destination folder.","Message",0)
```

```
<var_oper>(vDirDest,"",SELECT_FOLDER,"Select Destination Folder","", "0")
<proc_call>(ExitOnCancel,)
<msgoff>

<msg>(100,100,"Folder copy is in progress. Please wait....","Message",0)
<dircopy>("vDirSource","vDirDest",0,0,3,,vFail,"%TEMP%\DirCopyFailuresReport.txt","-pr -on ")
<msgoff>

<#> ----------------------------------------------------------------------

<proc_def_begin>(ExitOnCancel,)
<if_str>("_vCanceled==1")
   <exitmacro>
<endif>
<proc_def_end>

<#> ----------------------------------------------------------------------
```

MOVE - < dirmove >() ... [Pro]

Folder MOVE
<dirmove>("Source","Destination",Subfolders,Unused,Retries,Variable for number of processed files,Variable for number of failures,"Log errors","Additional options")
Available in: Professional edition

Moves directory including all subdirectories.

#	Parameter name	Parameter description
1	Source	Full path to the source directory (e.g., "c:\mydocuments").
2	Destination	Full path to the destination directory (e.g., "c:\newdocs").
3	Subfolders	Must be 0.
4	Unused	Must be 0.
5	Retries	
6	Variable for number of processed files	
7	Variable for number of failures	
8	Log errors	
9	Additional options	

Example (Macro Steps):

```
1   <#> <#> This macro moves directory you select to other directory you select.
2   Macro execution: ONLY COMMANDS
3   Message SHOW "" : "Select folder you want to move." (other parameters: x = 100, y = 100, Window title =
    Message, Buttons = None, Timeout (seconds) = , Always on top = ).
4   Variable OPERATION "SELECT_FOLDER" (Variable for result = vDirSource, Input text/variable = , Parameter
    1 = Select Source Folder, Parameter 2 = , Parameter 3 = 0)
5   Procedure CALL: ExitOnCancel with parameters ()
6   Message CLOSE
7   Message SHOW "" : "Select destination folder." (other parameters: x = 100, y = 100, Window title =
    Message, Buttons = None, Timeout (seconds) = , Always on top = ).
8   Variable OPERATION "SELECT_FOLDER" (Variable for result = vDirDest, Input text/variable = , Parameter 1
    = Select Destination Folder, Parameter 2 = , Parameter 3 = 0)
9   Procedure CALL: ExitOnCancel with parameters ()
10  Message CLOSE
11  Message SHOW "" : "Directory move is in progress. Please wait...." (other parameters: x = 100, y = 100,
    Window title = Message, Buttons = None, Timeout (seconds) = , Always on top = ).
12  Folder MOVE from "vDirSource" to "vDirDest" (Subfolders = No, Retries = , Variable for number of processed
    files = , Variable for number of failures = , Log errors = , Additional options = )
13  Message CLOSE
14  <#> <#> ------------------------------------------------------------
15  Procedure BEGIN: ExitOnCancel with parameters ()
16      IF STRING _vCanceled==1
17          Macro EXIT
18      ENDIF
19  Procedure END
20  <#> <#> ------------------------------------------------------------
```

Example (Plain Text):

```
<#> This macro moves directory you select to other directory you select.
<#>
<cmds>

<msg>(100,100,"Select folder you want to move.","Message",0)
<var_oper>(vDirSource,"",SELECT_FOLDER,"Select Source Folder","", "0")
<proc_call>(ExitOnCancel,)
<msgoff>

<msg>(100,100,"Select destination folder.","Message",0)
<var_oper>(vDirDest,"",SELECT_FOLDER,"Select Destination Folder","", "0")
```

```
<proc_call>(ExitOnCancel,)
<msgoff>

<msg>(100,100,"Directory move is in progress. Please wait....","Message",0)
<dirmove>("vDirSource","vDirDest",0,0)
<msgoff>

<#> ---------------------------------------------------------------------

<proc_def_begin>(ExitOnCancel,)
<if_str>("_vCanceled==1")
   <exitmacro>
<endif>
<proc_def_end>

<#> ---------------------------------------------------------------------
```

Recycle bin EMPTY - < recbinempty > ... [Pro]

Folder Recycle bin EMPTY
<recbinempty>
Available in: Professional edition

This command clears the content of the Windows recycle bin.

Example (Macro Steps):

1	**<#> <#> This example clears recycle bin**
2	**Folder Recycle bin EMPTY**

Example (Plain Text):

```
<#> This example clears recycle bin
<recbinempty>
```

RENAME - < dirrename >() ... [Pro]

Folder RENAME
<dirrename>("Folder","New name",Unused,Unused)
Available in: Professional edition

Renames directory specified.

#	Parameter name	Parameter description
1	Folder	(Full) path to the source directory (e.g., "c:\mydocuments").
2	New name	New name of the directory (for example, "mydocuments_old").
3	Unused	Must be 0.
4	Unused	Must be 0.

Example (Macro Steps):

```
1   <#> <#> This macro renames directory you select.
2   Macro execution: ONLY COMMANDS
3   Message SHOW "" : "Select folder you want to rename." (other parameters: x = 100, y = 100, Window title = Message, Buttons = None, Timeout (seconds) = , Always on top = ).
4   Variable OPERATION "SELECT_FOLDER" (Variable for result = vDirSource, Input text/variable = , Parameter 1 = Select Source Folder, Parameter 2 = , Parameter 3 = 0)
5   Procedure CALL: ExitOnCancel with parameters ()
6   Message CLOSE
7   Folder RENAME vDirSource, New name = RENAMED,Unused = 0
8   <#> <#> ----------------------------------------------------------------
9   Procedure BEGIN: ExitOnCancel with parameters ()
10      IF STRING _vCanceled==1
11          Macro EXIT
12      ENDIF
13  Procedure END
14  <#> <#> ----------------------------------------------------------------
```

Example (Plain Text):

```
<#> This macro renames directory you select.
<#>
<cmds>
```

```
<msg>(100,100,"Select folder you want to rename.","Message",0)
<var_oper>(vDirSource,"",SELECT_FOLDER,"Select Source Folder","", "0")
<proc_call>(ExitOnCancel,)
<msgoff>

<dirrename>("vDirSource","RENAMED",0,0)

<#> ----------------------------------------------------------------------

<proc_def_begin>(ExitOnCancel,)
<if_str>("_vCanceled==1")
   <exitmacro>
<endif>
<proc_def_end>

<#> ----------------------------------------------------------------------
```

ENCRYPT/DECRYPT - < dir_encryption >() ... [Pro]

Folder ENCRYPT/DECRYPT
<dir_encryption>("Input","Output",Operation,Encryption bit strength,"Password")
Available in: Professional edition

This command encrypts/decrypts all files within an input directory (folder) including all sub-folders. Encrypted files are located in output directory. An extension ".aes" is added to the encrypted files. For example, file "Invoice_01256.doc" is encrypted to "Invoice_01256.doc.aes".

#	Parameter name	Parameter description
1	Input	(Full) path to a directory (folder) that is to be encrypted/decrypted. Example: "c:\myPrivateDocuments".
2	Output	Directory (folder) where encrypted/decrypted files are located. Example: "c:\myPrivateDocuments_Encrypted".
3	Operation	This parameter can be one of these: ENCRYPT_AES - the files are encrypted using AES. DECRYPT_AES - the files (previously encrypted using ENCRYPT_AES) are decrypted.
4	Encryption bit strength	Strength of the encryption. These values are supported: 128, 192, 256.
5	Password	User defined password that is used to encrypt/decrypt the files. A directory encrypted by a password can be successfully decrypted again only if the same password is used.

Example (Macro Steps):

```
1   <#> <#> This example shows how to encrypt whole folder.
2   <#> <#> The second example shows how to decrypt it again.
3   Macro execution: ONLY COMMANDS
4   <#> <#> Create folder for sample files
5   Folder CREATE "c:\temp\dir_encryption_sample"
6   <#> <#> Create sample files
7   File SAVE TEXT "This is a sample file 1." to file "c:\temp\dir_encryption_sample\1.file_original.txt
8   File SAVE TEXT "This is a sample file 2." to file "c:\temp\dir_encryption_sample\2.file_original.txt
9   Folder ENCRYPT/DECRYPT Input = "c:\temp\dir_encryption_sample", Output = "
    c:\temp\dir_encryption_sample_Out", Operation = " ENCRYPT_AES", Encryption bit strength = "128"
10  <#> <#> Open folder to see the results
11  Folder OPEN open folder "c:\temp\dir_encryption_sample_Out" in Windows Explorer.
```

Example (Plain Text):

```
<#> This example shows how to encrypt whole folder.
```

<#> The second example shows how to decrypt it again.
<cmds>

<#> Create folder for sample files
<dircreate>("c:\temp\dir_encryption_sample",0)

<#> Create sample files
<data_save>("This is a sample file 1.","c:\temp\dir_encryption_sample\1.file_original.txt","")
<data_save>("This is a sample file 2.","c:\temp\dir_encryption_sample\2.file_original.txt","")

<dir_encryption>("c:\temp\dir_encryption_sample", "c:\temp\dir_encryption_sample_Out", ENCRYPT_AES,128,"dirpwd")

<#> Open folder to see the results
<diropen>("c:\temp\dir_encryption_sample_Out",0)

Keyboard

Key EXTENDED - < extkey > ... [Pro]

Keyboard Key EXTENDED
<extkey>
Available in: Professional edition

This command tells that following key (a 'key' command) is an extended key.

Example (Macro Steps):

1 <#> <#>This macro has the same effect as hitting 'Enter' key on numeric pad

2 **Keyboard Key EXTENDED**

3 **Key** Enter

Example (Plain Text):

<#>This macro has the same effect as hitting 'Enter' key on numeric pad
<extkey><enter>

Insert NEW LINE - < newline > ... [Pro]

Keyboard Insert NEW LINE
<newline>
Available in: Professional edition

This command inserts a new line (sends "Enter" keystroke to the currently active application).

Example (Macro Steps):

1	**<#> <#> This macro inserts new line**
2	**Keyboard Insert NEW LINE**

Example (Plain Text):

<#> This macro inserts new line
<newline>

BLOCK - < keys_block > ... [Pro]

Keyboard BLOCK
<keys_block>
Available in: Professional edition

This command blocks keyboard keys. It can be used when it is necessary to disable keyboard input during macro execution, for example, before "wait for key" command ("waitfor"). If it is required to disable keyboard and mouse input during whole macro execution then it is also possible to use "Lock keyboard and mouse while macro is running" option in the macro settings tab. To unblock keyboard, use "keys_unblock" command.

Example (Macro Steps):

1 <#> <#> This macro shows how to use 'keys_block' and 'keys_unblock' commands

2 **Macro execution: ONLY COMMANDS**

3 **Message SHOW** "Information" : "Waiting untill 'a' key is pressed...." (other parameters: x = 32, y = 32, Window title = Message, Buttons = None, Timeout (seconds) = 0, Always on top = No).

4 **Keyboard BLOCK**

5 **WAIT FOR** Object = "KEY", Event = "PRESS", Parameter = "a", Timeout (seconds) = "50", Exact = "0"

6 **WAIT FOR** Object = "KEY", Event = "RELEASE", Parameter = "a", Timeout (seconds) = "50", Exact = "0"

7 **Keyboard UNBLOCK**

8 **Message CLOSE**

Example (Plain Text):

```
<#> This macro shows how to use 'keys_block' and 'keys_unblock' commands
<cmds>
<msg>(32,32,"Waiting untill 'a' key is pressed....","Message",0,0,0,0)
<keys_block>
<waitfor>("KEY","PRESS","a",50,0)
<waitfor>("KEY","RELEASE","a",50,0)
<keys_unblock>
<msgoff>
```

UNBLOCK - < keys_unblock > ... [Pro]

Keyboard UNBLOCK
<keys_unblock>
Available in: Professional edition

This command unblock keyboard keys after a previous use of "keys_block" command.

Example (Macro Steps):

1 <#> <#> This macro shows how to use 'keys_block' and 'keys_unblock' commands

2 **Macro execution: ONLY COMMANDS**

3 **Message SHOW** "Information" : "Waiting untill 'a' key is pressed...." (other parameters: x = 32, y = 32, Window title = Message, Buttons = None, Timeout (seconds) = 0, Always on top = No).

4 **Keyboard BLOCK**

5 **WAIT FOR** Object = "KEY", Event = "PRESS", Parameter = "a", Timeout (seconds) = "50", Exact = "0"

6 **WAIT FOR** Object = "KEY", Event = "RELEASE", Parameter = "a", Timeout (seconds) = "50", Exact = "0"

7 **Keyboard UNBLOCK**

8 **Message CLOSE**

Example (Plain Text):

```
<#> This macro shows how to use 'keys_block' and 'keys_unblock' commands
<cmds>
<msg>(32,32,"Waiting untill 'a' key is pressed....","Message",0,0,0,0)
<keys_block>
<waitfor>("KEY","PRESS","a",50,0)
<waitfor>("KEY","RELEASE","a",50,0)
<keys_unblock>
<msgoff>
```

Key UP - < key_up >() ... [Free]

Keyboard Key UP
<key_up>(Keyboard key,Extended)
Available in: Free edition

The "key_down" and "key_up" commands allows to send a keystroke to currently active window (target application).
Note: For inserting a text it is simpler to use "free text".

#	Parameter name	Parameter description
1	Keyboard key	The key to press/release such as A, B, 1, /, etc. For Shift, Ctrl, F1, etc., keys <shift>, <ctrl>, <F1> syntax needs to be used. In addition, the KC:XXX key code can be used (Note: The KC:XXX key code is showing in the main window in lower right area as keys are being hit).
2	Extended	If 1, the extended key pressed/released. Otherwise have to be 0.

Example (Macro Steps):

1 <#> <#> This command presses down "A" key and releases it one second later

2 **Macro execution: ONLY COMMANDS**

3 **Keyboard Key DOWN** Keyboard key=< shift>, Extended=No

4 **Keyboard Key DOWN** Keyboard key=A, Extended=No

5 **WAIT** wait "1000" ms (time is constant: "")

6 **Keyboard Key UP** Keyboard key=A, Extended=No

7 **Keyboard Key UP** Keyboard key=< shift>, Extended=No

Example (Plain Text):

<#> This command presses down "A" key and releases it one second later
<cmds>
<key_down>(<shift>,0)
<key_down>(A,0)
<wx>(1000)
<key_up>(A,0)
<key_up>(<shift>,0)

Key DOWN - < key_down >() ... [Free]

Keyboard Key DOWN
<key_down>(Keyboard key,Extended)
Available in: Free edition

The "key_down" and "key_up" commands allows to send a keystroke to currently active window (target application).
Note: For inserting a text it is simpler to use "free text".

#	Parameter name	Parameter description
1	Keyboard key	The key to press/release such as A, B, 1, /, etc. For Shift, Ctrl, F1, etc., keys <shift>, <ctrl>, <F1> syntax needs to be used. In addition, the KC:XXX key code can be used (Note: The KC:XXX key code is showing in the main window in lower right area as keys are being hit).
2	Extended	If 1, the extended key pressed/released. Otherwise have to be 0.

Example (Macro Steps):

1 <#> <#> This command presses down "A" key and releases it one second later

2 **Macro execution: ONLY COMMANDS**

3 **Keyboard Key DOWN** Keyboard key=< shift>, Extended=No

4 **Keyboard Key DOWN** Keyboard key=A, Extended=No

5 **WAIT** wait "1000" ms (time is constant: "")

6 **Keyboard Key UP** Keyboard key=A, Extended=No

7 **Keyboard Key UP** Keyboard key=< shift>, Extended=No

Example (Plain Text):

```
<#> This command presses down "A" key and releases it one second later
<cmds>
<key_down>(<shift>,0)
<key_down>(A,0)
<wx>(1000)
<key_up>(A,0)
<key_up>(<shift>,0)
```

ScrollLock ON - < ScrollLock_ON > ... [Pro]

Keyboard ScrollLock ON
<ScrollLock_ON>
Available in: Professional edition

This command turns ScrollLock key ON.
Note: "_vlsScrollLockON" system variable can be used to determine state of the ScrollLock key.

Example (Macro Steps):

1	**<#> <#>This macro turns ScrollLock key ON**
2	**Keyboard ScrollLock ON**

Example (Plain Text):

<#>This macro turns ScrollLock key ON
<ScrollLock_ON>

ScrollLock OFF - < ScrollLock_OFF > ... [Pro]

Keyboard ScrollLock OFF
<ScrollLock_OFF>
Available in: Professional edition

This command turns ScrollLock key OFF.
Note: "_vlsScrollLockON" system variable can be used to determine state of the ScrollLock key.

Example (Macro Steps):

1	**<#> <#>This macro turns ScrollLock key OFF**
2	**Keyboard ScrollLock OFF**

Example (Plain Text):

<#>This macro turns ScrollLock key OFF
<ScrollLock_OFF>

CapsLock ON - < CapsLock_ON > ... [Pro]

Keyboard CapsLock ON
<CapsLock_ON>
Available in: Professional edition

This command turns CapsLock key ON.
Note: "_vIsCapsLockON" system variable can be used to determine state of the CapsLock key.

Example (Macro Steps):

```
1    <#> <#>This macro toggles CapsLock
2    Macro execution: ONLY COMMANDS
3    IF %_vIsCapsLockON%==YES
4        Keyboard CapsLock OFF
5    ELSE activate
6        Keyboard CapsLock ON
7    ENDIF
```

Example (Plain Text):

```
<#>This macro toggles CapsLock
<cmds>
<if>("%_vIsCapsLockON%==YES")
  <CapsLock_OFF>
<else>
  <CapsLock_ON>
<endif>
```

CapsLock OFF - < CapsLock_OFF > ... [Pro]

Keyboard CapsLock OFF
<CapsLock_OFF>
Available in: Professional edition

This command turns CapsLock key OFF.
Note: "_vIsCapsLockON" system variable can be used to determine state of the CapsLock key.

Example (Macro Steps):

```
1    <#> <#>This macro toggles CapsLock
2    Macro execution: ONLY COMMANDS
3    IF %_vIsCapsLockON%==YES
4        Keyboard CapsLock OFF
5    ELSE activate
6        Keyboard CapsLock ON
7    ENDIF
```

Example (Plain Text):

```
<#>This macro toggles CapsLock
<cmds>
<if>("%_vIsCapsLockON%==YES")
  <CapsLock_OFF>
<else>
  <CapsLock_ON>
<endif>
```

NumLock ON - < NumLock_ON > ... [Pro]

Keyboard NumLock ON
<NumLock_ON>
Available in: Professional edition

This command turns NumLock key ON.
Note: "_vIsNumLockON" system variable can be used to determine state of the NumLock key.

Example (Macro Steps):

```
1    <#> <#>This macro toggles NumLock
2    Macro execution: ONLY COMMANDS
3    IF %_vIsNumLockON%==YES
4        Keyboard NumLock OFF
5    ELSE activate
6        Keyboard NumLock ON
7    ENDIF
```

Example (Plain Text):

```
<#>This macro toggles NumLock
<cmds>
<if>("%_vIsNumLockON%==YES")
  <NumLock_OFF>
<else>
  <NumLock_ON>
<endif>
```

NumLock OFF - < NumLock_OFF > ... [Pro]

Keyboard NumLock OFF
<NumLock_OFF>
Available in: Professional edition

This command turns NumLock key OFF.
Note: "_vIsNumLockON" system variable can be used to determine state of the NumLock key.

Example (Macro Steps):

```
1    <#> <#>This macro toggles NumLock
2    Macro execution: ONLY COMMANDS
3    IF %_vIsNumLockON%==YES
4        Keyboard NumLock OFF
5    ELSE activate
6        Keyboard NumLock ON
7    ENDIF
```

Example (Plain Text):

```
<#>This macro toggles NumLock
<cmds>
<if>("%_vIsNumLockON%==YES")
  <NumLock_OFF>
<else>
  <NumLock_ON>
<endif>
```

SEND KEYSTROKES - < keystrokes >() ... [Pro]

Keyboard SEND KEYSTROKES
<keystrokes>("Keystrokes sequence","Delay (ms)","Keystrokes per delay")
Available in: Professional edition

The command sends defined sequence of keystrokes to the currently active window. The command allows to control the speed of the keystrokes by specifying two additional parameters. If they are set to 5 and 25 then the meaning is "delay sending of keystrokes for 5 milliseconds after each 25 keystrokes are sent".

#	Parameter name	Parameter description
1	Keystrokes sequence	The key to press/release such as a, b, 1, /, etc. For Shift, Ctrl, F1, etc., keys <shift>, <ctrl>, <F1> syntax needs to be used.
2	Delay (ms)	Time in milliseconds to wait after "keystrokes per delay" were executed.
3	Keystrokes per delay	Number of keystrokes to send before delay takes place.

Example (Macro Steps):

```
1   <#> <#> This sample slowly types 'Hello!' to Notepad (if it is open)
2   Macro execution: ONLY COMMANDS
3   IF WINDOW "[* - Notepad|Notepad|#46|#118]" Is Open (Match=Partial)
4       Window ACTIVATE bring "[* - Notepad|Notepad|#46|#118]" window to top (other parameters: Match =
        Partial, Window state = Normal, %p4_name = )
5       Keyboard SEND KEYSTROKES "Hello!", Delay (ms)="500"
6   ELSE activate
7       Message SHOW "" : "'Notepad' is not opened!" (other parameters: x = 100, y = 100, Window title =
        Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
8   ENDIF
```

Example (Plain Text):

```
<#> This sample slowly types 'Hello!' to Notepad (if it is open)
<cmds>

<if_win>("[* - Notepad|Notepad|#46|#118]","OPEN",0)
  <actwin>("[* - Notepad|Notepad|#46|#118]",0,0)
  <keystrokes>("Hello!","500","5")
<else>
  <msg>(100,100,"'Notepad' is not opened!","Message",1)
<endif>
```

Keys

- < numpad4 > ... [Free]

Key
<numpad4>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < numpad5 > ... [Free]

Key
<numpad5>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < numpad6 > ... [Free]

Key
<numpad6>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < numpad7 > ... [Free]

Key
<numpad7>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < numpad8 > ... [Free]

Key
<numpad8>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < numpad9 > ... [Free]

Key
<numpad9>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < numpad* > ... [Free]

Key
<numpad*>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < numpad+ > ... [Free]

Key
<numpad+>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < numpad- > ... [Free]

Key
<numpad->
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < numpad. > ... [Free]

Key
<numpad.>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < numpad/ > ... [Free]

Key
<numpad/>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F1 > ... [Free]

Key
<F1>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F2 > ... [Free]

Key
<F2>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F3 > ... [Free]

Key
<F3>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F4 > ... [Free]

Key
<F4>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F5 > ... [Free]

Key
<F5>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F6 > ... [Free]

Key
<F6>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F7 > ... [Free]

Key
<F7>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F8 > ... [Free]

Key
<F8>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < clear > ... [Free]

Key
<clear>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F9 > ... [Free]

Key
<F9>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F10 > ... [Free]

Key
<F10>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F11 > ... [Free]

Key
<F11>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F12 > ... [Free]

Key
<F12>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F13 > ... [Free]

Key
<F13>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F14 > ... [Free]

Key
<F14>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F15 > ... [Free]

Key
<F15>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F16 > ... [Free]

Key
<F16>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F17 > ... [Free]

Key
<F17>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F18 > ... [Free]

Key
<F18>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < enter > ... [Free]

Key
<enter>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F19 > ... [Free]

Key
<F19>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F20 > ... [Free]

Key
<F20>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F21 > ... [Free]

Key
<F21>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F22 > ... [Free]

Key
<F22>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F23 > ... [Free]

Key
<F23>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < F24 > ... [Free]

Key
<F24>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>0

- < scroll > ... [Free]

Key
<scroll>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < numlock > ... [Free]

Key
<numlock>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < shift > ... [Free]

Key
<shift>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < browser_back > ... [Free]

Key
<browser_back>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < browser_forward > ... [Free]

Key
<browser_forward>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < browser_refresh > ... [Free]

Key
<browser_refresh>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < browser_stop > ... [Free]

Key
<browser_stop>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < ctrl > ... [Free]

Key
<ctrl>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < browser_search > ... [Free]

Key
<browser_search>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < browser_favorites > ... [Free]

Key
<browser_favorites>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < browser_home > ... [Free]

Key
<browser_home>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < volume_mute > ... [Free]

Key
<volume_mute>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < volume_down > ... [Free]

Key
<volume_down>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < volume_up > ... [Free]

Key
<volume_up>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < media_nexttrack > ... [Free]

Key
<media_nexttrack>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < media_prevtrack > ... [Free]

Key
<media_prevtrack>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < media_stop > ... [Free]

Key
<media_stop>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < media_play_pause > ... [Free]

Key
<media_play_pause>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < alt > ... [Free]

Key
<alt>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < launch_mail > ... [Free]

Key
<launch_mail>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < launch_media_select > ... [Free]

Key
<launch_media_select>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < launch_app1 > ... [Free]

Key
<launch_app1>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < launch_app2 > ... [Free]

Key
<launch_app2>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	<#> <#> This will press F10 key
2	**Key** F10
3	<#> <#>This will press Ctrl+Alt+K
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < break > ... [Free]

Key
<break>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < capslock > ... [Free]

Key
<capslock>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < ctrld > ... [Free]

Key
<ctrld>
Available in: Free edition

This command cause Ctrl key is pressed down.

Example (Macro Steps):

1	<#> <#> This example shows how to simulate Ctrl+C key combination
2	**Key** Ctrl Down
3	**c**
4	**Key** Ctrl Up

Example (Plain Text):

<#> This example shows how to simulate Ctrl+C key combination
<ctrld>c<ctrlu>

- < ctrlu > ... [Free]

Key
<ctrlu>
Available in: Free edition

This command cause Ctrl key is released up.

Example (Macro Steps):

1	<#> <#> This example shows how to simulate Ctrl+C key combination
2	**Key** Ctrl Down
3	**c**
4	**Key** Ctrl Up

Example (Plain Text):

<#> This example shows how to simulate Ctrl+C key combination
<ctrld>c<ctrlu>

- < altd > ... [Free]

Key
<altd>
Available in: Free edition

This command cause Alt key is pressed down.

Example (Macro Steps):

1	<#> <#> This example shows how to simulate Alt+F4 key combination
2	**Key** Alt Down
3	**Key** F4
4	**Key** Alt Up

Example (Plain Text):

```
<#> This example shows how to simulate Alt+F4 key combination
<altd><F4><altu>
```

- < altu > ... [Free]

Key
<altu>
Available in: Free edition

This command cause Alt key is released up.

Example (Macro Steps):

1	<#> <#> This example shows how to simulate Alt+F4 key combination
2	**Key** Alt Down
3	**Key** F4
4	**Key** Alt Up

Example (Plain Text):

<#> This example shows how to simulate Alt+F4 key combination
<altd><F4><altu>

- < altd_r > ... [Free]

Key
<altd_r>
Available in: Free edition

This command cause right site Alt key is pressed down.

Example (Macro Steps):

1	<#> <#> This example shows how to simulate Alt+F4 key combination
2	**Key** Alt (right) Down
3	**Key** F4
4	**Key** Alt (right) Up

Example (Plain Text):

```
<#> This example shows how to simulate Alt+F4 key combination
<altd_r><F4><altu_r>
```

- < altu_r > ... [Free]

Key
<altu_r>
Available in: Free edition

This command cause right site Alt key is released up.

Example (Macro Steps):

1	<#> <#> This example shows how to simulate Alt+F4 key combination
2	**Key** Alt (right) Down
3	**Key** F4
4	**Key** Alt (right) Up

Example (Plain Text):

```
<#> This example shows how to simulate Alt+F4 key combination
<altd_r><F4><altu_r>
```

- < shiftd > ... [Free]

Key
<shiftd>
Available in: Free edition

This command cause Shift key is pressed down.

Example (Macro Steps):

1 **<#> <#> This example shows how to simulate Shift+a to type capital A**

2 **Key** Shift Down

3 **a**

4 **Key** Shift Up

Example (Plain Text):

<#> This example shows how to simulate Shift+a to type capital A
<shiftd>a<shiftu>

- < shiftu > ... [Free]

Key
<shiftu>
Available in: Free edition

This command cause Shift key is released up.

Example (Macro Steps):

1	**<#> <#> This example shows how to simulate Shift+a to type capital A**
2	**Key** Shift Down
3	**a**
4	**Key** Shift Up

Example (Plain Text):

<#> This example shows how to simulate Shift+a to type capital A
<shiftd>a<shiftu>

- < winkeyd > ... [Free]

Key
<winkeyd>
Available in: Free edition

This command causes Win key is pressed down.

Example (Macro Steps):

1	<#> <#> This example shows how to simulate Win key + M to minimize all windows
2	**Key** Win Down
3	**a**
4	**Key** Win (right) Down

Example (Plain Text):

<#> This example shows how to simulate Win key + M to minimize all windows
<winkeyd>a<winkeyu>

- < winkeyd_r > ... [Free]

Key
<winkeyd_r>
Available in: Free edition

This command causes Win key is released up.

Example (Macro Steps):

1	<#> <#> This example shows how to simulate Win key + M to minimize all windows
2	**Key** Win Down
3	**a**
4	**Key** Win (right) Down

Example (Plain Text):

<#> This example shows how to simulate Win key + M to minimize all windows
<winkeyd>a<winkeyu>

- < winkeyu > ... [Free]

Key
<winkeyu>
Available in: Free edition

This command causes right site Win key is pressed down.

Example (Macro Steps):

1	<#> <#> This example shows how to simulate Win key + M to minimize all windows
2	**Key** Win Up
3	**a**
4	**Key** Win (right) Up

Example (Plain Text):

<#> This example shows how to simulate Win key + M to minimize all windows
<winkeyd_r>a<winkeyu_r>

- < winkeyu_r > ... [Free]

Key
<winkeyu_r>
Available in: Free edition

This command causes right site Win key is released up.

Example (Macro Steps):

1	**<#> <#> This example shows how to simulate Win key + M to minimize all windows**
2	**Key** Win Up
3	**a**
4	**Key** Win (right) Up

Example (Plain Text):

<#> This example shows how to simulate Win key + M to minimize all windows
<winkeyd_r>a<winkeyu_r>

- < esc > ... [Free]

Key
<esc>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < space > ... [Free]

Key
<space>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < pgup > ... [Free]

Key
<pgup>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < pgdn > ... [Free]

Key
<pgdn>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < end > ... [Free]

Key
<end>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < home > ... [Free]

Key
<home>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < left > ... [Free]

Key
<left>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < up > ... [Free]

Key
<up>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < right > ... [Free]

Key
<right>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < down > ... [Free]

Key
<down>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < select > ... [Free]

Key
<select>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < execkey > ... [Free]

Key
<execkey>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < printscreen > ... [Free]

Key
<printscreen>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < insert > ... [Free]

Key
<insert>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < delete > ... [Free]

Key
<delete>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < back > ... [Free]

Key
<back>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < tab > ... [Free]

Key
<tab>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < lwinkey > ... [Free]

Key
<lwinkey>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < rwinkey > ... [Free]

Key
<rwinkey>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < appskey > ... [Free]

Key
<appskey>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < numpad0 > ... [Free]

Key
<numpad0>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < numpad1 > ... [Free]

Key
<numpad1>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < numpad2 > ... [Free]

Key
<numpad2>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

- < numpad3 > ... [Free]

Key
<numpad3>
Available in: Free edition

The keystroke is sent to the curently active application (window).

Notice: The control keys - **<ctrl>, <alt>, <shift>, <lwinkey>, <rwinkey>** - behaves so that the first occurence means "key down" and the second occurence means "key up". This is different from all other keys where the key means "key up and down".

Example (Macro Steps):

1	**<#> <#> This will press F10 key**
2	**Key** F10
3	**<#> <#>This will press Ctrl+Alt+K**
4	**Key** Ctrl
5	**Key** Alt
6	**k**
7	**Key** Alt
8	**Key** Ctrl

Example (Plain Text):

<#> This will press F10 key
<F10><#>This will press Ctrl+Alt+K
<ctrl><alt>k<alt><ctrl>

Macro Engine

Macro CHANGE ICON - < me_changeicon >() ... [Pro]

Macro CHANGE ICON
<me_changeicon>("Macro","Icon file",Icon index)
Available in: Professional edition

This command changes a macro icon.

#	Parameter name	Parameter description
1	Macro	Name of the macro. The macro with this name must exist otherwise the command fails.
2	Icon file	Full path to a file containing icon (exe, dll, ico, etc.).
3	Icon index	Index of the icon within the IconFilePath.

Example (Macro Steps):

1 <#> <#>This macro changes icon of macro named "M1"

2 **Macro CHANGE ICON** Macro=M1, Icon file=notepad.exe, Icon index=0

Example (Plain Text):

```
<#>This macro changes icon of macro named "M1"
<me_changeicon>("M1","notepad.exe",0)
```

Macro execution: ONLY COMMANDS - < cmds > ... [Free]

Macro execution: ONLY COMMANDS
<cmds>
Available in: Free edition

This command changes how the macro is played back. The part of the macro that follows after this command is played back so that only commands are executed while other macro text (keys) is ignored. This prevents macro from inserting unwanted keys (for example new lines used just to format macro) during playback. To enable playback of all macro text (keys) use "keys" command.

Example (Macro Steps):

1 <#> <#> This macro shows how to use 'cmds' and 'keys' commands

2 **Macro execution: ONLY COMMANDS**

3 **Jump TARGET** "lbl_Again"

4 **Message SHOW** "" : "Starting Notepad ..." (other parameters: x = 100, y = 100, Window title = Message, Buttons = None, Timeout (seconds) = , Always on top =).

5 **Run APPLICATION** "notepad.exe" (other parameters: Parameters = , Folder path = , Window state = Normal). Macro execution waits for application to finish up to "0" seconds.

6 **WAIT FOR** Object = "WIN", Event = "OPEN", Parameter = "[* - Notepad|Notepad|#0|#119]", Timeout (seconds) = "15", Exact = "0"

7 **Message CLOSE**

8 **Macro execution: KEYS / FREE TEXT + COMMANDS**

9 **Hello, \ \ this macro just demonstrates 'cmds/keys' commands. **

10 **Macro execution: ONLY COMMANDS**

11 **Variable SET** "vAgain=YES/NO", Message text="Do you want to run this macro again ?"

12 **IF STRING** vAgain==YES

13 **Jump TO** "lbl_Again"

14 **ENDIF**

Example (Plain Text):

```
<#> This macro shows how to use 'cmds' and 'keys' commands
<cmds>

<label>("lbl_Again")
<msg>(100,100,"Starting Notepad ...","Message",0)
<execappex>("notepad.exe","","",0,0)
<waitfor>("WIN","OPEN","[* - Notepad|Notepad|#0|#119]",15,0)
```

```
<msgoff>

<keys>Hello,

this macro just demonstrates 'cmds/keys' commands.
<cmds>

<varset>("vAgain=YES/NO","Do you want to run this macro again ?")
<if_str>("vAgain==YES")
   <goto>("lbl_Again")
<endif>
```

Macro execution: KEYS / FREE TEXT + COMMANDS - < keys > ... [Free]

Macro execution: KEYS / FREE TEXT + COMMANDS
<keys>
Available in: Free edition

This command changes how the macro is played back. The part of the macro that follows after this command is played back so that whole macro text (keys) are executed. Use this command to enable playback of all macro text (keys) if "cmds" command was previously used.

Example (Macro Steps):

1 <#> <#> This macro shows how to use 'cmds' and 'keys' commands

2 **Macro execution: ONLY COMMANDS**

3 **Jump TARGET** "lbl_Again"

4 **Message SHOW** "" : "Starting Notepad ..." (other parameters: x = 100, y = 100, Window title = Message, Buttons = None, Timeout (seconds) = , Always on top =).

5 **Run APPLICATION** "notepad.exe" (other parameters: Parameters = , Folder path = , Window state = Normal). Macro execution waits for application to finish up to "0" seconds.

6 **WAIT FOR** Object = "WIN", Event = "OPEN", Parameter = "[* - Notepad|Notepad|#0|#119]", Timeout (seconds) = "15", Exact = "0"

7 **Message CLOSE**

8 **Macro execution: KEYS / FREE TEXT + COMMANDS**

9 **Hello, \ \ this macro just demonstrates 'cmds/keys' commands. **

10 **Macro execution: ONLY COMMANDS**

11 **Variable SET** "vAgain=YES/NO", Message text="Do you want to run this macro again ?"

12 **IF STRING** vAgain==YES

13 **Jump TO** "lbl_Again"

14 **ENDIF**

Example (Plain Text):

```
<#> This macro shows how to use 'cmds' and 'keys' commands
<cmds>

<label>("lbl_Again")
<msg>(100,100,"Starting Notepad ...","Message",0)
<execappex>("notepad.exe","","",0,0)
<waitfor>("WIN","OPEN","[* - Notepad|Notepad|#0|#119]",15,0)
<msgoff>
```

```
<keys>Hello,

this macro just demonstrates 'cmds/keys' commands.
<cmds>

<varset>("vAgain=YES/NO","Do you want to run this macro again ?")
<if_str>("vAgain==YES")
   <goto>("lbl_Again")
<endif>
```

Macro ENABLE/DISABLE - < me_macroenable >() ... [Pro]

Macro ENABLE/DISABLE
<me_macroenable>("Macro",Option,Unused)
Available in: Professional edition

Enables or disables macro.

#	Parameter name	Parameter description
1	Macro	Name of the macro to enable/disable. The macro with this name must exist otherwise the command fails.
2	Option	0 - disable 1 - enable
3	Unused	Must be 0.

Example (Macro Steps):

1 <#> <#> This macro will disable 'TestMacro1' (if it exist)

2 **Macro execution: ONLY COMMANDS**

3 **Macro ENABLE/DISABLE** "TestMacro1", Option="Disable"

Example (Plain Text):

```
<#> This macro will disable 'TestMacro1' (if it exist)
<#>
<cmds>
<me_macroenable>("TestMacro1",0,0)
```

Macro group ENABLE/DISABLE - < me_macroenable_group >() ... [Pro]

Macro group ENABLE/DISABLE
<me_macroenable_group>("Macro group",Option,Unused)
Available in: Professional edition

Enables or disables specified macro group.

#	Parameter name	Parameter description
1	Macro group	Name of the macro group to enable/disable. The macro group with this name must exist otherwise the command fails.
2	Option	0 - disable 1 - enable
3	Unused	Must be 0.

Example (Macro Steps):

1 <#> <#> This macro will disable all macros from 'GRP1' group (if exists)

2 **Macro execution: ONLY COMMANDS**

3 **Macro group ENABLE/DISABLE** "GRP1", Option="Disable"

Example (Plain Text):

```
<#> This macro will disable all macros from 'GRP1' group (if exists)
<#>
<cmds>
<me_macroenable_group>("GRP1",0,0)
```

Macro program EXIT - < me_exit >() ... [Pro]

Macro program EXIT
<me_exit>(Option)
Available in: Professional edition

This command exits/restarts this program.

#	Parameter name	Parameter description
1	Option	0 - exit this program 1 - restart the program

Example (Macro Steps):

1 <#> <#> This macro will restart program

2 **Macro execution: ONLY COMMANDS**

3 **Macro program EXIT** Option=Restart

Example (Plain Text):

```
<#> This macro will restart program
<#>
<cmds>
<me_exit>(1)
```

Macro execution: DISABLE "Shift+Esc" hotkey. - < me_stop_disable > ... [Pro]

Macro execution: DISABLE "Shift+Esc" hotkey.
<me_stop_disable>
Available in: Professional edition

Macro execution can be (by default) stopped using "Shift+Esc" hotkey. This command disables this option. After this command is processed, pressing "Shift+Esc" will have no effect. The "me_stop_enable" command enables "Shift+Esc" again.

Example (Macro Steps):

1 <#> <#> This macro shows how to enable/disable "Shift+Esc"

2 **Macro execution: ONLY COMMANDS**

3 **Macro execution: DISABLE "Shift+Esc" hotkey.**

4 <#> <#> Part of the macro that cannot be interrupted for integrity reasons comes here

5 <#> <#> We will supply it by 5 seconds wait command in this example

6 **Message SHOW** "" : "You cannot stop macro execution using "Shift+Esc" hotkey now. Wait for 5 seconds, please..." (other parameters: x = -100, y = -100, Window title = Message, Buttons = None, Timeout (seconds) = , Always on top =).

7 **WAIT** wait "5000" ms (time is constant: "")

8 **Message CLOSE**

9 **Macro execution: ENABLE "Shift+Esc" hotkey.**

10 <#> <#> The rest of the macro can be interrupted using "Shift+Esc"

11 **Message SHOW** "" : "Now you can stop macro execution using "Shift+Esc" hotkey." (other parameters: x = -100, y = -100, Window title = Message, Buttons = None, Timeout (seconds) = , Always on top =).

12 **WAIT** wait "5000" ms (time is constant: "")

13 **Message CLOSE**

Example (Plain Text):

```
<#> This macro shows how to enable/disable "Shift+Esc"
<#>
<cmds>
<me_stop_disable>
<#> Part of the macro that cannot be interrupted for integrity reasons comes here
<#> We will supply it by 5 seconds wait command in this example
<msg>(-100,-100,"You cannot stop macro execution using %_vQuoteChar%Shift+Esc%_vQuoteChar% hotkey now.
Wait for 5 seconds, please...","Message",0)
<wx>(5000)
```

```
<msgoff>
<me_stop_enable>
<#> The rest of the macro can be interrupted using "Shift+Esc"
<msg>(-100,-100,"Now you can stop macro execution using %_vQuoteChar%Shift+Esc%_vQuoteChar%
hotkey.","Message",0)
<wx>(5000)
<msgoff>
```

Macro execution: ENABLE "Shift+Esc" hotkey. - < me_stop_enable > ... [Pro]

Macro execution: ENABLE "Shift+Esc" hotkey.
<me_stop_enable>
Available in: Professional edition

This command enables stopping of macro execution by "Shift+Esc" hotkey. This command is typically used after disables this option. After this command is processed, pressing "Shift+Esc" will have no effect. The "me_stop_enable" command enables "Shift+Esc" again.

Example (Macro Steps):

1 <#> <#> This macro shows how to enable/disable "Shift+Esc"

2 **Macro execution: ONLY COMMANDS**

3 **Macro execution: DISABLE "Shift+Esc" hotkey.**

4 <#> <#> Part of the macro that cannot be interrupted for integrity reasons comes here

5 <#> <#> We will supply it by 5 seconds wait command in this example

6 **Message SHOW** "" : "You cannot stop macro execution using "Shift+Esc" hotkey now. Wait for 5 seconds, please..." (other parameters: x = -100, y = -100, Window title = Message, Buttons = None, Timeout (seconds) = , Always on top =).

7 **WAIT** wait "5000" ms (time is constant: "")

8 **Message CLOSE**

9 **Macro execution: ENABLE "Shift+Esc" hotkey.**

10 <#> <#> The rest of the macro can be interrupted using "Shift+Esc"

11 **Message SHOW** "" : "Now you can stop macro execution using "Shift+Esc" hotkey." (other parameters: x = -100, y = -100, Window title = Message, Buttons = None, Timeout (seconds) = , Always on top =).

12 **WAIT** wait "5000" ms (time is constant: "")

13 **Message CLOSE**

Example (Plain Text):

```
<#> This macro shows how to enable/disable "Shift+Esc"
<#>
<cmds>
<me_stop_disable>
<#> Part of the macro that cannot be interrupted for integrity reasons comes here
<#> We will supply it by 5 seconds wait command in this example
<msg>(-100,-100,"You cannot stop macro execution using %_vQuoteChar%Shift+Esc%_vQuoteChar% hotkey now.
Wait for 5 seconds, please...","Message",0)
<wx>(5000)
```

```
<msgoff>
<me_stop_enable>
<#> The rest of the macro can be interrupted using "Shift+Esc"
<msg>(-100,-100,"Now you can stop macro execution using %_vQuoteChar%Shift+Esc%_vQuoteChar%
hotkey.","Message",0)
<wx>(5000)
<msgoff>
```

Macro execution STATUS WINDOW - < me_status_window >() ... [Pro]

Macro execution STATUS WINDOW

<me_status_window>("Window title",Operation,Not always on top,x,y,Width,Height)
Available in: Professional edition

This command allows to show/close a window that displays macro execution status. There can be only one status window shown at the same time. The window can contain multiple rows containing icon (none, "in progress", "OK" "Failed") and status text. The "me_status_set" command allows to add/modify content of the status window - it either adds new row or modifies existing row.

#	Parameter name	Parameter description
1	Window title	Title of the status window.
2	Operation	0 - the status window is to be opened 1 - the status window is to be closed
3	Not always on top	0 - normal window that can be overlapped by other windows 1 - always on top window that is never overlapped by other windows
4	x	X - position of the window on the screen (absolute screen coordinates).
5	y	Y - position of the window on the screen (absolute screen coordinates).
6	Width	Width of the window.
7	Height	Height of the window.

Example (Macro Steps):

```
1   <#> <#> This sample shows how to us the status window
2   Macro execution: ONLY COMMANDS
3   Macro execution STATUS WINDOW "Macro in progres" (Operation = OPEN, x =
    EXPR(%_vMonitorWorkAreaX_1%+%_vMonitorWorkAreaCX_1%-350), y =
    EXPR(%_vMonitorWorkAreaY_1%+%_vMonitorWorkAreaCY_1%-150), Width = 350, Height = 150)
4   Macro execution: STATUS UPDATE Item identifier=0, Item name=Three steps needs to be done, please
    wait:, Status icon=None
5   Macro execution: STATUS UPDATE Item identifier=1, Item name=Step 1, Status icon=In progress
6   WAIT wait "2000" ms (time is constant: "")
7   Macro execution: STATUS UPDATE Item identifier=1, Item name=Step 1: OK, Status icon=OK
8   Macro execution: STATUS UPDATE Item identifier=2, Item name=Step 2, Status icon=In progress
9   WAIT wait "2000" ms (time is constant: "")
10  Macro execution: STATUS UPDATE Item identifier=2, Item name=Step 2: Failed, Status icon=Failure
11  Macro execution: STATUS UPDATE Item identifier=3, Item name=Step 3, Status icon=In progress
12  WAIT wait "2000" ms (time is constant: "")
13  Macro execution: STATUS UPDATE Item identifier=3, Item name=Step 3: OK, Status icon=OK
14  WAIT wait "2000" ms (time is constant: "")
```

Example (Plain Text):

```
<#> This sample shows how to us the status window
<cmds>

<me_status_window>("Macro in
progres",0,1,EXPR(%_vMonitorWorkAreaX_1%+%_vMonitorWorkAreaCX_1%-350),EXPR(%_vMonitorWorkAreaY_1%+%_v
MonitorWorkAreaCY_1%-150),350,150)
<me_status_set>(0,"Three steps needs to be done, please wait:",0)

<me_status_set>(1,"Step 1",1)
<wx>(2000)
<me_status_set>(1,"Step 1: OK",2)

<me_status_set>(2,"Step 2",1)
<wx>(2000)
<me_status_set>(2,"Step 2: Failed",3)

<me_status_set>(3,"Step 3",1)
<wx>(2000)
<me_status_set>(3,"Step 3: OK",2)

<wx>(2000)
```

Macro execution: STATUS UPDATE - < me_status_set >() ... [Pro]

Macro execution: STATUS UPDATE
<me_status_set>(Item identifier,"Item name",Status icon)
Available in: Professional edition

This command adds or modifies row in the status window ("me_status_window" command). Each row is identified using numeric ID. If there is not a row with the given ID in the window yet, then new row is added. If a row with the given ID exists, then it is updated using parameters passed (status and text - if the text is not supplied then old text remains displayed). Each row consists of status icon (none, "in progress", "OK", "Failed") and text.

#	Parameter name	Parameter description
1	Item identifier	Unique numeric identifier of the row in the status window.
2	Item name	Text showing on the row.
3	Status icon	0 - none icon 1 - "in progress" icon 2 - "OK" icon 3 - "Failed" icon 4 - "Info" icon

Example (Macro Steps):

1 <#> <#> This sample shows how to us the status window

2 **Macro execution: ONLY COMMANDS**

3 **Macro execution STATUS WINDOW** "Macro in progres" (Operation = OPEN, x = 100, y = 100, Width = 350, Height = 150)

4 **Macro execution: STATUS UPDATE** Item identifier=0, Item name=Three steps needs to be done, please wait:, Status icon=None

5 **Macro execution: STATUS UPDATE** Item identifier=1, Item name=Step 1, Status icon=In progress

6 **WAIT** wait "2000" ms (time is constant: "")

7 **Macro execution: STATUS UPDATE** Item identifier=1, Item name=Step 1: OK, Status icon=OK

8 **Macro execution: STATUS UPDATE** Item identifier=2, Item name=Step 2, Status icon=In progress

9 **WAIT** wait "2000" ms (time is constant: "")

10 **Macro execution: STATUS UPDATE** Item identifier=2, Item name=Step 2: Failed, Status icon=Failure

11 **Macro execution: STATUS UPDATE** Item identifier=3, Item name=Step 3, Status icon=In progress

12 **WAIT** wait "2000" ms (time is constant: "")

13 **Macro execution: STATUS UPDATE** Item identifier=3, Item name=Step 3: OK, Status icon=OK

14 **WAIT** wait "2000" ms (time is constant: "")

Example (Plain Text):

```
<#> This sample shows how to us the status window
<cmds>

<me_status_window>("Macro in progres",0,1,100,100,350,150)
<me_status_set>(0,"Three steps needs to be done, please wait:",0)

<me_status_set>(1,"Step 1",1)
<wx>(2000)
<me_status_set>(1,"Step 1: OK",2)

<me_status_set>(2,"Step 2",1)
<wx>(2000)
<me_status_set>(2,"Step 2: Failed",3)

<me_status_set>(3,"Step 3",1)
<wx>(2000)
<me_status_set>(3,"Step 3: OK",2)

<wx>(2000)
```

Macro execution: Progress/Cancel SHOW - < me_macroprogress_show > ... [Pro]

Macro execution: Progress/Cancel SHOW
<me_macroprogress_show>
Available in: Professional edition

This command shows macro execution progress window with Cancel/Pause button.

Example (Macro Steps):

```
1   <#> <#> This macro shows/hides macro progress execution window
2   Macro execution: ONLY COMMANDS
3   Macro execution: Progress/Cancel SHOW
4   WAIT wait "3000" ms (time is constant: "")
5   <#> <#> Wait 3 seconds
6   Macro execution: Progress/Cancel HIDE
7   WAIT wait "3000" ms (time is constant: "")
8   <#> <#> Wait 3 seconds
9   Macro execution: Progress/Cancel SHOW
10  WAIT wait "3000" ms (time is constant: "")
11  <#> <#> Wait 3 seconds
```

Example (Plain Text):

```
<#> This macro shows/hides macro progress execution window
<#>
<cmds>
<me_macroprogress_show>
<wx>(3000) <#> Wait 3 seconds
<me_macroprogress_hide>
<wx>(3000) <#> Wait 3 seconds
<me_macroprogress_show>
<wx>(3000) <#> Wait 3 seconds
```

Macro execution: Progress/Cancel HIDE - < me_macroprogress_hide > ... [Pro]

Macro execution: Progress/Cancel HIDE
<me_macroprogress_hide>
Available in: Professional edition

This command hides macro execution progress window with Cancel/Pause button.

Example (Macro Steps):

1 <#> <#> This macro shows/hides macro progress execution window
2 **Macro execution: ONLY COMMANDS**
3 **Macro execution: Progress/Cancel SHOW**
4 **WAIT** wait "3000" ms (time is constant: "")
5 <#> <#> Wait 3 seconds
6 **Macro execution: Progress/Cancel HIDE**
7 **WAIT** wait "3000" ms (time is constant: "")
8 <#> <#> Wait 3 seconds
9 **Macro execution: Progress/Cancel SHOW**
10 **WAIT** wait "3000" ms (time is constant: "")
11 <#> <#> Wait 3 seconds

Example (Plain Text):

```
<#> This macro shows/hides macro progress execution window
<#>
<cmds>
<me_macroprogress_show>
<wx>(3000) <#> Wait 3 seconds
<me_macroprogress_hide>
<wx>(3000) <#> Wait 3 seconds
<me_macroprogress_show>
<wx>(3000) <#> Wait 3 seconds
```

Macro File: Set dirty - < me_setfiledirty >() ... [Pro]

Macro File: Set dirty
<me_setfiledirty>(Option)
Available in: Professional edition

This command sets the macro file internal "changed" flag ON/OFF. If the flag is ON then the program attempts to save macro file (either automatically or asking user).

#	Parameter name	Parameter description
1	Option	Can be one of these values: 0 - the macro file is not changed (and will not be saved) 1 - the macro file is changed (and wil be saved)

Example (Macro Steps):

1 <#> <#>This macro resets the macro file chnage flag

2 **Macro File: Set dirty** "No"

Example (Plain Text):

<#>This macro resets the macro file chnage flag
<me_setfiledirty>(0)

Macro Flow Control

PAUSE - < pause > ... [Pro]

PAUSE
<pause>
Available in: Professional edition

Stops (pauses) macro execution. User needs to keep Ctrl key down for 0.5 seconds to let the macro to continue.

Example (Macro Steps):

1 <#> <#> This macro shows how to use 'pause' command

2 **Macro execution: ONLY COMMANDS**

3 **Message SHOW** "" : "Start Notepad and then press 'Ctrl' key for 0.5 seconds to continue this macro." (other parameters: x = 100, y = 100, Window title = Message, Buttons = None, Timeout (seconds) = , Always on top =).

4 **PAUSE** pause macro execution until user hits Enter key

5 **Message CLOSE**

6 **Window ACTIVATE** bring "notepad" window to top (other parameters: Match = Partial, Window state = Normal, %p4_name = no)

7 **Macro execution: KEYS / FREE TEXT + COMMANDS**

8 **Dear Mr.**

9 **Message SHOW** "" : "Insert the customer name and then press 'Ctrl' key for 0.5 seconds to continue this macro." (other parameters: x = 100, y = 100, Window title = Message, Buttons = None, Timeout (seconds) = , Always on top =).

10 **PAUSE** pause macro execution until user hits Enter key

11 **Message CLOSE**

12 **, \ \ we are writing you this letter because...**

Example (Plain Text):

```
<#> This macro shows how to use 'pause' command
<#>
<cmds>
<msg>(100,100,"Start Notepad and then press 'Ctrl' key for 0.5 seconds to continue this macro.","Message",0)
<pause>
<msgoff>
<actwin>("notepad",0,0,"no")
<keys>Dear Mr. <msg>(100,100,"Insert the customer name and then press 'Ctrl' key for 0.5 seconds to continue this
macro.","Message",0)<pause><msgoff>,

we are writing you this letter because...
```

WAIT - < wx >() ... [Free]

WAIT
<wx>(Time to wait,Wait time is constant (not adjusted to macro speed))
Available in: Free edition

The command postpones macro execution for X milliseconds. The behavior of this command depends on the macro playback speed that can user specify for each macro in the "Item Properties" dialog box.

#	Parameter name	Parameter description
1	Time to wait	Time to wait in milliseconds.
2	Wait time is constant (not adjusted to macro speed)	Wait time is constant (not adjusted to macro speed).

Example (Macro Steps):

1 <#> <#> This macro shows how to use 'wx' command

2 **Macro execution: ONLY COMMANDS**

3 **Variable SET** "vTime=", Message text="How many seconds to wait ?"

4 <#> <#> Calculate time to wait from seconds to milliseconds

5 **Variable OPERATION** "CALC_EXPRESSION" (Variable for result = vTime, Input text/variable = %vTime%*1000, Parameter 1 = 0, Parameter 2 = , Parameter 3 = 0)

6 **WAIT** wait "vTime" ms (time is constant: "")

7 **Message SHOW** "" : "Specified time is out." (other parameters: x = 100, y = 100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

Example (Plain Text):

```
<#> This macro shows how to use 'wx' command
<#>
<cmds>
<varset>("vTime=","How many seconds to wait ?")
<#> Calculate time to wait from seconds to milliseconds
<var_oper>(vTime,"%vTime%*1000",CALC_EXPRESSION,"0","", "0")
<wx>(vTime)
<msg>(100,100,"Specified time is out.","Message",1)
```

Loop BEGIN - < begloop >() ... [Pro]

Loop BEGIN
<begloop>(Repeat)
Available in: Professional edition

Macro loop feature allows to repeat several times a part of macro (macro steps) that are enclosed between "begin loop" and "end loop" commands.

#	Parameter name	Parameter description
1	Repeat	The number of repeats. Can be one of these options: Repeat > 0 - The Repeat is the actual number of loops to be performed. Repeat = 0 - The user specifies the number of repeats in the runtime. Repeat = -1 - The user is asked each loop to continue or stop repeating.

Example (Macro Steps):

1 <#> <#> This example shows how to use loop commands

2 **Macro execution: ONLY COMMANDS**

3 **Variable SET** "vNum=", Message text="How many Notepads you want to open ?"

4 **Loop BEGIN** Repeat = vNum

5 **Run APPLICATION** "notepad.exe" (other parameters: Parameters = , Folder path = , Window state = Normal). Macro execution waits for application to finish up to "0" seconds.

6 **Loop END**

Example (Plain Text):

```
<#> This example shows how to use loop commands
<#>
<cmds>
<varset>("vNum=","How many Notepads you want to open ?")
<begloop>(vNum)
  <execappex>("notepad.exe","","",0,0)
<endloop>
```

Loop END - < endloop > ... [Pro]

Loop END
<endloop>
Available in: Professional edition

Begins loop. The loop allows to repeat some part of macro many times. Each commands must be followed by .

Example (Macro Steps):

1 <#> <#> This example shows how to use loop commands

2 **Macro execution: ONLY COMMANDS**

3 **Variable SET** "vNum=", Message text="How many Notepads you want to open ?"

4 **Loop BEGIN** Repeat = vNum

5 **Run APPLICATION** "notepad.exe" (other parameters: Parameters = , Folder path = , Window state = Normal). Macro execution waits for application to finish up to "0" seconds.

6 **Loop END**

Example (Plain Text):

```
<#> This example shows how to use loop commands
<#>
<cmds>
<varset>("vNum=","How many Notepads you want to open ?")
<begloop>(vNum)
  <execappex>("notepad.exe","","",0,0)
<endloop>
```

IF - < if >() ... [Pro]

IF
<if>("expression")
Available in: Professional edition

The command evaluates the expression and if it is evaluated as "true" then following macro steps (steps between "if" and "else/endif") are executed. The expression can contain EXPR(...) syntax.

#	Parameter name	Parameter description
1	expression	Expression can contain brackets () and these operators are available: == (is equal) != (is not equal) <= (is smalller or equal) < (is smaller) >= (is bigger or equal) > (is bigger) ~= (contains substring) _AND_ (the condition is true AND also following condition is true) _OR_ (the condition is tru OR the following condition is true) Expression examples: Var1==Var2 10020) _AND_ (%vText%~=substring)

Example (Macro Steps):

1 <#> <#> This example shows how to use "if" command.

2 **Macro execution: ONLY COMMANDS**

3 **Variable SET** "vNum=", Message text="Type a number:"

4 **IF** 100<(30+%vNum%)

5 **Message SHOW** "Information" : "Yes, this is true: 100<30+%vNum%" (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

6 **ELSE** activate

7 **Message SHOW** "Error" : "No, it is not true: 100<30+%vNum%" (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

8 **ENDIF**

Example (Plain Text):

```
<#> This example shows how to use "if" command.
<#>
<cmds>
<varset>("vNum=","Type a number:")
<if>("100<(30+%vNum%)")
<msg>(-100,-100,"Yes, this is true: 100<30+%vNum%","Message",1,,0)
<else>
<msg>(-100,-100,"No, it is not true: 100<30+%vNum%","Message",1,,2)
```

<endif>

ELSE - < else > ... [Free]

ELSE
<else>
Available in: Free edition

The command begins an optional block of macro steps. The block is executed if the associated "if" command expression is evaluated as "false". The command can be only use together with one of an "if" command.

Example (Macro Steps):

```
1   <#> <#> This example shows how to use "if" command.
2   Macro execution: ONLY COMMANDS
3   Variable SET "vNum=", Message text="Type a number:"
4   IF 100<(30+%vNum%)
5       Message SHOW "Information" : "Yes, this is true: 100<30+%vNum%" (other parameters: x = -100, y =
        -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
6   ELSE activate
7       Message SHOW "Error" : "No, it is not true: 100<30+%vNum%" (other parameters: x = -100, y = -100,
        Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
8   ENDIF
```

Example (Plain Text):

```
<#> This example shows how to use "if" command.
<#>
<cmds>
<varset>("vNum=","Type a number:")
<if>("100<(30+%vNum%)")
<msg>(-100,-100,"Yes, this is true: 100<30+%vNum%","Message",1,,0)
<else>
<msg>(-100,-100,"No, it is not true: 100<30+%vNum%","Message",1,,2)
<endif>
```

ENDIF - < endif > ... [Free]

ENDIF
<endif>
Available in: Free edition

The command closes conditional block of macro steps. The command can be only use together with one of an "if" command.

Example (Macro Steps):

```
1   <#> <#> This example shows how to use "if" command.
2   Macro execution: ONLY COMMANDS
3   Variable SET "vNum=", Message text="Type a number:"
4   IF 100<(30+%vNum%)
5       Message SHOW "Information" : "Yes, this is true: 100<30+%vNum%" (other parameters: x = -100, y =
        -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
6   ELSE activate
7       Message SHOW "Error" : "No, it is not true: 100<30+%vNum%" (other parameters: x = -100, y = -100,
        Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
8   ENDIF
```

Example (Plain Text):

```
<#> This example shows how to use "if" command.
<#>
<cmds>
<varset>("vNum=","Type a number:")
<if>("100<(30+%vNum%)")
<msg>(-100,-100,"Yes, this is true: 100<30+%vNum%","Message",1,,0)
<else>
<msg>(-100,-100,"No, it is not true: 100<30+%vNum%","Message",1,,2)
<endif>
```

Send KEYSTROKES as FAST as possible - < faston > ... [Pro]

Send KEYSTROKES as FAST as possible
<faston>
Available in: Professional edition

The command changes the speed of keystrokes sending to the highest rate. Macro keystrokes are sent to active application without any delay.

Example (Macro Steps):

1 <#> <#> This macro shows how to use 'faston/fastoff' command.

2 **Run APPLICATION** "notepad.exe" (other parameters: Parameters = , Folder path = , Window state = Normal). Macro execution waits for application to finish up to "0" seconds.

3 **WAIT FOR** Object = "WIN", Event = "ACT", Parameter = "Notepad", Timeout (seconds) = "10", Exact = "0"

4 **Send KEYSTROKES on SLOWEST rate**

5 **This text is being inserted slower... **

6 **Send KEYSTROKES as FAST as possible**

7 **...and this text is being inserted faster.**

Example (Plain Text):

```
<#> This macro shows how to use 'faston/fastoff' command.
<#>
<execappex>("notepad.exe","","",0,0)<waitfor>("WIN","ACT","Notepad",10,0)<fastoff>This text is being inserted slower...
<faston>...and this text is being inserted faster.
```

Send KEYSTROKES on SLOWEST rate - < fastoff > ... [Pro]

Send KEYSTROKES on SLOWEST rate
<fastoff>
Available in: Professional edition

The command changes the speed of keystrokes sending to the slowest rate. Macro keystrokes are sent to active application with a short delay in between them.

Example (Macro Steps):

1 <#> <#> This macro shows how to use 'faston/fastoff' command.

2 **Run APPLICATION** "notepad.exe" (other parameters: Parameters = , Folder path = , Window state = Normal). Macro execution waits for application to finish up to "0" seconds.

3 **WAIT FOR** Object = "WIN", Event = "ACT", Parameter = "Notepad", Timeout (seconds) = "10", Exact = "0"

4 **Send KEYSTROKES on SLOWEST rate**

5 **This text is being inserted slower... **

6 **Send KEYSTROKES as FAST as possible**

7 **...and this text is being inserted faster.**

Example (Plain Text):

```
<#> This macro shows how to use 'faston/fastoff' command.
<#>
<execappex>("notepad.exe","","",0,0)<waitfor>("WIN","ACT","Notepad",10,0)<fastoff>This text is being inserted slower...
<faston>...and this text is being inserted faster.
```

Jump TARGET - < label >() ... [Pro]

Jump TARGET
<label>("Label")
Available in: Professional edition

Specifies the step in the macro where you can jump using "Jump TO" ("goto") command.

#	Parameter name	Parameter description
1	Label	Name of label that marks the step to jump to.

Example (Macro Steps):

```
1   <#> <#> This macro shows how to jump to particular step in the macro
2   Macro execution: ONLY COMMANDS
3   Loop BEGIN Repeat = 0
4       IF NUMERIC _vLoopCounter>5
5           Message SHOW "" : "Too many loops, jumping to the end." (other parameters: x = 100, y = 100,
            Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
6           Jump TO "END"
7       ENDIF
8   Loop END
9   Message SHOW "" : "OK" (other parameters: x = 100, y = 100, Window title = Message, Buttons = OK,
    Timeout (seconds) = , Always on top = ).
10  Jump TARGET "END"
```

Example (Plain Text):

```
<#> This macro shows how to jump to particular step in the macro
<#>
<cmds>
<begloop>(0)
<if_num>("_vLoopCounter>5")
  <msg>(100,100,"Too many loops, jumping to the end.","Message",1)
  <goto>("END")
<endif>
<endloop>

<msg>(100,100,"OK","Message",1)
<label>("END")
```

Jump TO - < goto >() ... [Pro]

Jump TO
<goto>("Label")
Available in: Professional edition

Jump to a step defined by Jump TARGET ("label") command in the macro and continue macro execution from this step.

#	Parameter name	Parameter description
1	Label	Name of the target step to jump to.

Example (Macro Steps):

```
1   <#> <#> This macro shows how to jump to defined step in the macro
2   Macro execution: ONLY COMMANDS
3   Variable SET "vNumOfLoops=", Message text="Insert number of loops (less than 25)"
4   IF NUMERIC vNumOfLoops>=25
5       Message SHOW "" : "Too many loops, jumping to the end." (other parameters: x = 100, y = 100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
6       Jump TO "END"
7   ENDIF
8   Loop BEGIN Repeat = vNumOfLoops
9       Message SHOW "" : "_vLoopCounter" (other parameters: x = -100, y = -100, Window title = Loop, Buttons = None, Timeout (seconds) = , Always on top = ).
10      WAIT wait "200" ms (time is constant: "")
11  Loop END
12  Jump TARGET "END"
```

Example (Plain Text):

```
<#> This macro shows how to jump to defined step in the macro
<#>
<cmds>
<varset>("vNumOfLoops=","Insert number of loops (less than 25)")
<if_num>("vNumOfLoops>=25")
  <msg>(100,100,"Too many loops, jumping to the end.","Message",1)
  <goto>("END")
<endif>
<begloop>(vNumOfLoops)
  <msg>(-100,-100,"_vLoopCounter","Loop",0)
  <wx>(200)
<endloop>
```

<label>("END")

Macro EXIT - < exitmacro > ... [Pro]

Macro EXIT
<exitmacro>
Available in: Professional edition

This command stops macro execution. If the macro is called using "run" command from other macro, also the calling macro is stopped.

Example (Macro Steps):

```
1    <#> <#> This macro shows how to use command
2    Macro execution: ONLY COMMANDS
3    Message SHOW "" : "Do you want to exit macro execution now?" (other parameters: x = -100, y = -100, Window title = Message, Buttons = Yes and No, Timeout (seconds) = , Always on top = ).
4    IF STRING _vMsgButton==YES
5        Macro EXIT
6    ENDIF
7    Message SHOW "" : "OK, macro execution continues then..." (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
```

Example (Plain Text):

```
<#> This macro shows how to use <exitmacro> command
<cmds>
<msg>(-100,-100,"Do you want to exit macro execution now?","Message",2)
<if_str>("_vMsgButton==YES")
<exitmacro>
<endif>
<msg>(-100,-100,"OK, macro execution continues then...","Message",1)
```

WAIT FOR - < waitfor >() ... [Free]

WAIT FOR
<waitfor>("Object","Event","Parameter",Timeout (seconds),Exact)
Available in: Free edition

Wait for a special event (window, clipboard, key, mouse) to occur.

#	Parameter name	Parameter description
1	Object	Can be one of these: "WIN" - a window related event is expected "CLIP" - a clipboard related event is expected "KEY" - a specific key press is expected "MOUSE" - a mouse click is expected "IMAGE_ON_SCREEN_EXACT" - an image on screen will appear/disappear, image must be exactly the same "IMAGE_ON_SCREEN_TOLERANT" - an image on screen will appear/disappear, image does not has to be exactly the same - some level of tolerance is allowed "WEBBROWSER" - Internet Explorer web browser is loading data "FILE" - a file is created or deleted "FOLDER" - a folder is created or deleted
2	Event	These events are supported for particular Object: "WIN" : - "OPEN" - wait until the window specified by Param is opened. - "CLOSE" - wait until the window specified by Param is closed. - "ACT" - wait until the window specified by Param is activated. - "NOACT" - wait until the window specified by Param is deactivated. "CLIP" : - "EMPTY" - wait until the clipboard is emptied. - "SET" - wait until some data are put into the clipboard. "KEY" : - "" (key is pressed) - wait until some of the keys specified by Param is pressed. "MOUSE" : - "" (mouse click) - wait until mouse click specified by Param appears. "IMAGE_ON_SCREEN_EXACT" - "APPEAR" - wait until the image appears on screens - "DISAPPEAR" - wait until the image disappears from screens "IMAGE_ON_SCREEN_TOLERANT" - "APPEAR" - wait until the image appears on screens - "DISAPPEAR" - wait until the image disappears from screens "WEBBROWSER" - "LOADING" - wait until the web page is fully loaded "FILE" - "CREATE" - wait until the file is created - "DELETE" - wait until the file is deleted - "CHANGED" - wait until the file is changed (based on the last write time) - "CAN_READ" - wait until the file can be open for reading (after it was previously blocked by some process) - "CAN_WRITE" - wait until the file can be open for writing (after it was previously blocked by some process) "FOLDER" - "CREATE" - wait until the folder is created - "DELETE" - wait until the folder is deleted
3	Parameter	Has this meaning depending on the Object: "WIN" - window title "CLIP" - not used "KEY" - can be one or more keys. For example, if Param is "abc<F6>KC:27<alt>" the "waitfor" waits until either a or b or c or F6 or ESC or Alt key is pressed. The KC:XXX is a key code number of a key on keyboard. The KC:XXX is showing in the main window in lower right area as keys are being hit. This way the user can know what is key code of each keyboard key. "MOUSE" - can be one of these: <mlbu> - left mouse button <mmbu> - middle mouse button <mrbu> - right mouse button "IMAGE_ON_SCREEN_EXACT" - file path to the .bmp file with the image. "IMAGE_ON_SCREEN_TOLERANT" - file path to the .bmp file with the image. "WEBBROWSER" - web page URL. "FILE" - (full) path to the file. "FOLDER" - (full) path to the file.

Example (Macro Steps):

1 <#> <#> This macro shows how to use 'waitfor' command

2 **Macro execution: ONLY COMMANDS**

3 **Message SHOW** "" : "Press 'Ctrl' key to continue this macro." (other parameters: x = 100, y = 100, Window title = Message, Buttons = None, Timeout (seconds) = , Always on top =).

4 **WAIT FOR** Object = "KEY", Event = "", Parameter = "", Timeout (seconds) = "5", Exact = "0"

5 **Message CLOSE**

6 **Message SHOW** "" : "Press left mouse button to continue this macro." (other parameters: x = 100, y = 100, Window title = Message, Buttons = None, Timeout (seconds) = , Always on top =).

7 **WAIT FOR** Object = "MOUSE", Event = "", Parameter = "", Timeout (seconds) = "5", Exact = "0"

8 **Message CLOSE**

9 **Message SHOW** "" : "Macro is finished." (other parameters: x = 100, y = 100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

Example (Plain Text):

```
<#> This macro shows how to use 'waitfor' command
<#>
<cmds>
<msg>(100,100,"Press 'Ctrl' key to continue this macro.","Message",0)
<waitfor>("KEY","","<ctrl>",5,0)
<msgoff>
<msg>(100,100,"Press left mouse button to continue this macro.","Message",0)
<waitfor>("MOUSE","","<mlbu>",5,0)
<msgoff>
<msg>(100,100,"Macro is finished.","Message",1)
```

IF WINDOW - < if_win >() ... [Free]

IF WINDOW
<if_win>("Window","Condition",Match)
Available in: Free edition

The command evaluates a window specific condition and if it is evaluated as "true" then following macro steps (steps between "if" and "else/endif") are executed.

#	Parameter name	Parameter description
1	Window	Window identification to check against the condition.
2	Condition	Condition can be one from the following: "OPEN" - the statement is true if window with the WinTitle is opened (exist). "CLOSE" - the statement is true if window with the WinTitle is not opened (no more exist). "ACT" - the statement is true if window with the WinTitle is active (top most, receives keyboard events). "NOACT" - the statement is true if window with the WinTitle is not active (not receives keyboard events).
3	Match	If 1, window identification must exactly match the window. If 0, window identification can just partially match the window.

Example (Macro Steps):

```
1   <#> <#> This macro shows how to use "if_win" command
2   Macro execution: ONLY COMMANDS
3   IF WINDOW "notepad" Is Open (Match=Partial)
4       IF WINDOW "Notepad" Is Active (Match=Partial)
5           Message SHOW "" : "Notepad is opened and active." (other parameters: x = 100, y = 100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
6       ELSE activate
7           Message SHOW "" : "Notepad is opened but not active." (other parameters: x = 100, y = 100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
8       ENDIF
9   ELSE activate
10      Message SHOW "" : "Notepad is not opened." (other parameters: x = 100, y = 100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
11  ENDIF
```

Example (Plain Text):

```
<#> This macro shows how to use "if_win" command
<#>
```

```
<cmds>
<if_win>("notepad","OPEN",0)
  <if_win>("Notepad","ACT",0)
    <msg>(100,100,"Notepad is opened and active.","Message",1)
  <else>
    <msg>(100,100,"Notepad is opened but not active.","Message",1)
  <endif>
<else>
  <msg>(100,100,"Notepad is not opened.","Message",1)
<endif>
```

IF FILE - < if_file >() ... [Pro]

IF FILE
<if_file>("File","Condition","Parameter")
Available in: Professional edition

The command evaluates a file specific condition and if it is evaluated as "true" then following macro steps (steps between "if" and "else/endif") are executed.

#	Parameter name	Parameter description
1	File	Full path to the file to check against the condition.
2	Condition	Condition can be one from the following: "EXIST" - the condition is true if the file exist. "NOTEXIST" - the condition is true if the file does not exist. "BIGGER" - the condition is true if the file size is bigger than the specified size. "SMALLER" - the condition is true if the file size is smaller than the specified size. "CONTAIN_NOCASE" - the condition is true if the file contains the text specified in the additional parameter. The text is compared "case insensitive". "CONTAIN_CASE" - the condition is true if the file contains the text specified in the additional parameter. The text is compared "case sensitive". "NOT_CONTAIN_NOCASE" - the condition is true if the file does not contain the text specified in the additional parameter. The text is compared "case insensitive". "NOT_CONTAIN_CASE" - the condition is true if the file does not contain the text specified in the additional parameter. The text is compared "case sensitive".
3	Parameter	Additional parameter: File size in bytes for BIGGER and SMALLER conditions. Text to find in the file for CONTAINS and NOT_CONTAINS conditions.

Example (Macro Steps):

1 <#> <#> This macro shows how to use if-file command

2 **Macro execution: ONLY COMMANDS**

3 **IF FILE** "somefile.txt" Exist (0)

4 **Message SHOW** "Information" : "Somefile.txt surprisingly exist!" (other parameters: x = 100, y = 100, Window title = Message, Buttons = OK, Timeout (seconds) = 0, Always on top = No).

5 **ELSE** activate

6 **Message SHOW** "Information" : "Somefile.txt file does not exist." (other parameters: x = 100, y = 100, Window title = Message, Buttons = OK, Timeout (seconds) = 0, Always on top = No).

7 **ENDIF**

Example (Plain Text):

```
<#> This macro shows how to use if-file command
<cmds>
<if_file>("somefile.txt","EXIST","0")
  <msg>(100,100,"Somefile.txt surprisingly exist!","Message",1,0,0,0)
```

```
<else>
   <msg>(100,100,"Somefile.txt file does not exist.","Message",1,0,0,0)
<endif>
```

IF FOLDER - < if_dir >() ... [Pro]

IF FOLDER
<if_dir>("Folder path","Condition","Unused")
Available in: Professional edition

The command evaluates a folder specific condition and if it is evaluated as "true" then following macro steps (steps between "if" and "else/endif") are executed.

#	Parameter name	Parameter description
1	Folder path	Full path to folder to check against the condition.
2	Condition	Condition can be one from the following: "EXIST" - the condition is true if the directory exist. "NOTEXIST" - the condition is true if the directory does not exist.
3	Unused	Must be empty.

Example (Macro Steps):

```
1   <#> <#> This macro shows how to use if-dir command
2   Macro execution: ONLY COMMANDS
3   IF FOLDER "c:\windows" Exist
4       Message SHOW "" : "'c:\windows' exist." (other parameters: x = 100, y = 100, Window title = Message,
        Buttons = OK, Timeout (seconds) = , Always on top = ).
5   ELSE activate
6       Message SHOW "" : "'c:\windows' NOT exist." (other parameters: x = 100, y = 100, Window title =
        Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
7   ENDIF
```

Example (Plain Text):

```
<#> This macro shows how to use if-dir command
<#>
<cmds>
<if_dir>("c:\windows","EXIST","")
  <msg>(100,100,"'c:\windows' exist.","Message",1)
<else>
  <msg>(100,100,"'c:\windows' NOT exist.","Message",1)
<endif>
```

IF CLIPBOARD - < if_clp >() ... [Pro]

IF CLIPBOARD
<if_clp>("Condition","Unused")
Available in: Professional edition

The command evaluates the clipboard specific condition and if it is evaluated as "true" then following macro steps (steps between "if" and "else/endif") are executed.

#	Parameter name	Parameter description
1	Condition	Condition can be one from the following: "EMPTY" - the condition is true if the clipboard is empty. "SET" - the condition is true if the clipboard contains some data.
2	Unused	Must be empty.

Example (Macro Steps):

```
1   <#> <#> This macro shows how to use if-clp command
2   Macro execution: ONLY COMMANDS
3   IF CLIPBOARD "Is Empty"
4       Message SHOW "" : "Clipboard is empty." (other parameters: x = 100, y = 100, Window title = Message,
        Buttons = OK, Timeout (seconds) = , Always on top = ).
5   ELSE activate
6       Variable SET "vEmpty=YES/NO", Message text="Clipboard is not empty. Do you want to empty it now?"
7       IF STRING vEmpty==YES
8           Clipboard CLEAR
9       ENDIF
10  ENDIF
```

Example (Plain Text):

```
<#> This macro shows how to use if-clp command
<#>
<cmds>
<if_clp>("EMPTY","")
  <msg>(100,100,"Clipboard is empty.","Message",1)
<else>
  <varset>("vEmpty=YES/NO","Clipboard is not empty. Do you want to empty it now?")
  <if_str>("vEmpty==YES")
    <clpempty>
  <endif>
<endif>
```

IF NUMERIC - < if_num >() ... [Pro]

IF NUMERIC
<if_num>("expression")
Available in: Professional edition

The command evaluates a numeric expression and if it is evaluated as "true" then following macro steps (steps between "if" and "else/endif") are executed. The expression can NOT contain EXPR(...) syntax.

#	Parameter name	Parameter description
1	expression	Expression can contain brackets () and these operators are available: == (is equal) != (is not equal) <= (is smalller or equal) < (is smaller) >= (is bigger or equal) > (is bigger) _AND_ (the condition is true AND also following condition is true) _OR_ (the condition is tru OR the following condition is true) Expression examples: %Var1%==%Var2% 100<%vNum%+10 (%x%>%y%) _OR_ (%x%+10<%y%)

Example (Macro Steps):

1 <#> <#> This example shows how to use "if" command.

2 **Macro execution: ONLY COMMANDS**

3 **Variable SET** "vNum=", Message text="Type a number:"

4 **IF NUMERIC** 100<(30+%vNum%)

5 **Message SHOW** "Information" : "Yes, this is true: 100<30+%vNum%" (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

6 **ELSE** activate

7 **Message SHOW** "Error" : "No, it is not true: 100<30+%vNum%" (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

8 **ENDIF**

Example (Plain Text):

```
<#> This example shows how to use "if" command.
<#>
<cmds>
<varset>("vNum=","Type a number:")
<if_num>("100<(30+%vNum%)")
<msg>(-100,-100,"Yes, this is true: 100<30+%vNum%","Message",1,,0)
<else>
<msg>(-100,-100,"No, it is not true: 100<30+%vNum%","Message",1,,2)
```

<endif>

IF STRING - < if_str >() ... [Pro]

IF STRING
<if_str>("expression")
Available in: Professional edition

The command evaluates a numeric expression and if it is evaluated as "true" then following macro steps (steps between "if" and "else/endif") are executed. All operands in the expression are considered to be strings (unlike "if_num" that consideres all operands to be numbers).

#	Parameter name	Parameter description
1	expression	Expression can contain brackets () and these operators are available: == (is equal) != (is not equal) <= < >= > ~= (contains substring) _AND_ (the condition is true AND also following condition is true) _OR_ (the condition is tru OR the following condition is true) Expression examples: %Var1%==%Var2% %str1%<%str2% %vText%~=substring (%str1%<%str2%) _AND_ (%vText%~=substring)

Example (Macro Steps):

```
1   <#> <#> This example shows how to use "if_str" command.
2   Macro execution: ONLY COMMANDS
3   Variable SET "v=", Message text="Type 'Hello' here"
4   IF STRING %v%==Hello
5       Message SHOW "Information" : "Yes! You really typed 'Hello'." (other parameters: x = -100, y = -100,
        Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
6   ELSE activate
7       Message SHOW "Error" : "No, you have typed something else...." (other parameters: x = -100, y = -100,
        Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
8   ENDIF
```

Example (Plain Text):

```
<#> This example shows how to use "if_str" command.
<cmds>
<varset>("v=","Type 'Hello' here")
<if_str>("%v%==Hello")
  <msg>(-100,-100,"Yes! You really typed 'Hello'.","Message",1,,0)
```

```
<else>
   <msg>(-100,-100,"No, you have typed something else....","Message",1,,2)
<endif>
```

Debug BREAK POINT - < -dbp- > ... [Pro]

Debug BREAK POINT
<-dbp->
Available in: Professional edition

Debugger build in this program allows user to walk through macro step-by-step. It is possible to let run macro in normal speed until it reaches "-dbp-" debug break point (there can be many of them within macro). The debug break point just stops macro execution in debugger. When macro is not running in debugging mode then this command has no effect.

Example (Macro Steps):

1 <#> <#> This macro shows how to use 'debug break point'

2 **Macro execution: ONLY COMMANDS**

3 **Message SHOW** "" : "Message 1" (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

4 **Message SHOW** "" : "Message 2" (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

5 **Debug BREAK POINT**

6 **Message SHOW** "" : "Message 3" (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

Example (Plain Text):

```
<#> This macro shows how to use 'debug break point'
<#>
<cmds>
<msg>(-100,-100,"Message 1","Message",1)
<msg>(-100,-100,"Message 2","Message",1)
<-dbp->
<msg>(-100,-100,"Message 3","Message",1)
```

If PROCESS - < if_process >() ... [Pro]

If PROCESS
<if_process>(File,Condition)
Available in: Professional edition

This command checks whether specified process is running or not. If the condition is evaluated as "true" then macro steps between "if" and "else/endif" are executed.

#	Parameter name	Parameter description
1	File	Name of the executable file (example: notepad.exe) or process identification number (example: 101876).
2	Condition	Must be one of these values: EXIST NOT_EXIST

Example (Macro Steps):

```
1   <#> <#> This example checks whether Notepad is running
2   <#> <#>
3   Macro execution: ONLY COMMANDS
4   If PROCESS "notepad.exe" Exist
5       Message SHOW "" : "Notepad is running." (other parameters: x = -100, y = -100, Window title = Message,
        Buttons = OK, Timeout (seconds) = , Always on top = ).
6   ELSE activate
7       Message SHOW "" : "Notepad is NOT running." (other parameters: x = -100, y = -100, Window title =
        Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
8   ENDIF
```

Example (Plain Text):

```
<#> This example checks whether Notepad is running
<#>
<cmds>
<if_process>(notepad.exe,EXIST)
  <msg>(-100,-100,"Notepad is running.","Message",1)
<else>
  <msg>(-100,-100,"Notepad is NOT running.","Message",1)
<endif>
```

Error CLEAR - < me_error_clear > ... [Pro]

Error CLEAR
<me_error_clear>
Available in: Professional edition

Errors that occur are saved in _vErr system variable. This variable can be used to check whether a macro command failed or not. This command clears the variable.

Example (Macro Steps):

```
1   <#> <#> This macro shows how to use 'me_error_nodisplay' and 'me_error_clear'
2   Macro execution: ONLY COMMANDS
3   Error message DISABLED
4   <#> <#> Try to get file size of non-existing file. Error
5   <#> <#> occurs but default (automatic) error message is replaced by the user defined one.
6   File INFO : save "Size" information of file "c:\notexistingfile.none" to variable "v1"
7   IF STRING _vErr != NO
8       Message SHOW "" : "Error! This is custom error message. The automatic error message was disabled by
        'me_err_nodisplay' command." (other parameters: x = -100, y = -100, Window title = Message, Buttons =
        OK, Timeout (seconds) = , Always on top = ).
9   ENDIF
10  Error CLEAR
11  IF STRING %_vErr%== NO
12      Message SHOW "Information" : "Error was cleared." (other parameters: x = -100, y = -100, Window title =
        Message, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
13  ENDIF
```

Example (Plain Text):

```
<#> This macro shows how to use 'me_error_nodisplay' and 'me_error_clear'
<cmds>
<me_error_nodisplay>
<#> Try to get file size of non-existing file. Error
<#> occurs but default (automatic) error message is replaced by the user defined one.
<fileinfo>("c:\notexistingfile.none","SIZE","v1")
<if_str>("_vErr != NO")
   <msg>(-100,-100,"Error!  This is custom error message.  The automatic error message was disabled by
'me_err_nodisplay' command.","Message",1)
<endif><#>
<me_error_clear><#>
<if_str>("%_vErr%== NO")<#>
```

<msg>(-100,-100,"Error was cleared.","Message",1,0,0,0)<#>
<endif>

Error message DISABLED - < me_error_nodisplay > ... [Pro]

Error message DISABLED
<me_error_nodisplay>
Available in: Professional edition

This command causes that if a command during macro execution fails then error message is not shown and macro execution continues.

Example (Macro Steps):

```
1   <#> <#> This macro shows how to use 'me_error_nodisplay'
2   Macro execution: ONLY COMMANDS
3   Error message DISABLED
4   <#> <#> Try to get file size of non-existing file. Error
5   <#> <#> occurs but default (automatic) error message is replaced by the user defined one.
6   File INFO : save "Size" information of file "c:\notexistingfile.none" to variable "v1"
7   IF STRING _vErr != NO
8       Message SHOW "" : "Error! This is custom error message. The automatic error message was disabled by
        'me_err_nodisplay' command." (other parameters: x = -100, y = -100, Window title = Message, Buttons =
        OK, Timeout (seconds) = , Always on top = ).
9   ENDIF
```

Example (Plain Text):

```
<#> This macro shows how to use 'me_error_nodisplay'
<#>
<cmds>
<me_error_nodisplay>
<#> Try to get file size of non-existing file. Error
<#> occurs but default (automatic) error message is replaced by the user defined one.
<fileinfo>("c:\notexistingfile.none","SIZE","v1")
<if_str>("_vErr != NO")
   <msg>(-100,-100,"Error!  This is custom error message.  The automatic error message was disabled by
'me_err_nodisplay' command.","Message",1)
<endif>
```

Error message ENABLED - < me_error_display > ... [Pro]

Error message ENABLED
<me_error_display>
Available in: Professional edition

This command causes that if a command during macro execution fails then error message is shown and macro execution is optionaly stopped. This is default behavior

Example (Macro Steps):

```
1    <#> <#> This macro shows how to use 'me_error_display'
2    Macro execution: ONLY COMMANDS
3    Error message ENABLED
4    <#> <#> Try to get file size of non-existing file. Error
5    <#> <#> occurs and error message is displayed
6    File INFO : save "Size" information of file "c:\notexistingfile.none" to variable "v1"
```

Example (Plain Text):

```
<#> This macro shows how to use 'me_error_display'
<#>
<cmds>
<me_error_display>
<#> Try to get file size of non-existing file. Error
<#> occurs and error message is displayed
<fileinfo>("c:\notexistingfile.none","SIZE","v1")
```

Macro EXIT (do not exit calling macro) - < exitmacro_soft > ... [Pro]

Macro EXIT (do not exit calling macro)
<exitmacro_soft>
Available in: Professional edition

This command stops macro execution. If the macro is called using "run" command from other macro then the calling macro is not stopped and continues execution.

Example (Macro Steps):

1 <#> <#> This macro shows how to use "exitmacro_soft command

2 **Macro execution: ONLY COMMANDS**

3 **Message SHOW** "" : "Do you want to exit macro execution now?" (other parameters: x = -100, y = -100, Window title = Message, Buttons = Yes and No, Timeout (seconds) = , Always on top =).

4 **IF STRING** _vMsgButton==YES

5 **Macro EXIT (do not exit calling macro)**

6 **ENDIF**

7 **Message SHOW** "" : "OK, macro execution continues then..." (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

Example (Plain Text):

```
<#> This macro shows how to use "exitmacro_soft command
<cmds>
<msg>(-100,-100,"Do you want to exit macro execution now?","Message",2)
<if_str>("_vMsgButton==YES")
<exitmacro_soft>
<endif>
<msg>(-100,-100,"OK, macro execution continues then...","Message",1)
```

If KEY / MOUSE BUTTON - < if_key >() ... [Pro]

If KEY / MOUSE BUTTON
<if_key>("Identifier","State")
Available in: Professional edition

This command is used to determine whether a keyboard key (mouse button) is pressed or not.

#	Parameter name	Parameter description
1	Identifier	a,b,c,d, etc. key or a special key syntax like , , etc or KC:XXX (key code, for example KC:27). The KC:XXX is a key code number of a key on keyboard. The KC:XXX is showing in the main window in lower right area as keys are being hit. This way the user can know what is key code of each keyboard key. As for the mouse buttons, one of the following identifier can be used: - left mouse button - middle mouse button - right mouse button
2	State	Can be either DOWN or UP

Example (Macro Steps):

1 <#> <#> If-keys command example

2 **Macro execution: ONLY COMMANDS**

3 **Message SHOW** "" : "Press 'F12' key to continue..." (other parameters: x = -100, y = -100, Window title = Message, Buttons = None, Timeout (seconds) = , Always on top =).

4 **Keyboard BLOCK**

5 **Jump TARGET** "loop"

6 **If KEY / MOUSE BUTTON** "" is "Down" then execute following steps

7 **Macro EXIT**

8 **ENDIF**

9 **Jump TO** "loop"

Example (Plain Text):

```
<#> If-keys command example
<cmds>
<msg>(-100,-100,"Press 'F12' key to continue...","Message",0)
<keys_block>
<label>("loop")
<if_key>("<F12>","DOWN")
<exitmacro>
<endif>
<goto>("loop")
```

Procedure END - < proc_def_end > ... [Pro]

Procedure END
<proc_def_end>
Available in: Professional edition

This command defines the end of procedure.

Example (Macro Steps):

```
1   <#> <#> This example shows how to use procedures
2   Macro execution: ONLY COMMANDS
3   Procedure BEGIN: AddQuotes with parameters (parStringInput, &parStringOutput&)
4       Variable SET "parStringOutput="%parStringInput%"", Message text=""
5   Procedure END
6   Procedure BEGIN: ConvertToUpper with parameters (parStringInput,&parStringOutput&)
7       Variable OPERATION "STR_UPPER" (Variable for result = parStringOutput, Input text/variable =
        %parStringInput%, Parameter 1 = 2, Parameter 2 = , Parameter 3 = 0)
8   Procedure END
9   Variable SET "vMyText=", Message text="Insert text you want to convert to upper case and enclose to
    quotes:"
10  IF STRING _vCanceled==1
11      Macro EXIT
12  ENDIF
13  Procedure CALL: ConvertToUpper with parameters (%vMyText%, vMyText )
14  Procedure CALL: AddQuotes with parameters (%vMyText%, vMyText )
15  Message SHOW "Information" : "%vMyText%" (other parameters: x = -100, y = -100, Window title = Result,
    Buttons = OK, Timeout (seconds) = 0, Always on top = No).
```

Example (Plain Text):

```
<#> This example shows how to use procedures
<cmds>
<proc_def_begin>(AddQuotes,"parStringInput", "&parStringOutput&")
  <varset>("parStringOutput=%_vQuoteChar%%parStringInput%%_vQuoteChar%","")
<proc_def_end>

<proc_def_begin>(ConvertToUpper,"parStringInput","&parStringOutput&")
  <var_oper>(parStringOutput,"%parStringInput%",STR_UPPER,"2","", "0")
```

```
<proc_def_end>

<varset>("vMyText=","Insert text you want to convert to upper case and enclose to quotes:")
<if_str>("_vCanceled==1")<#>
<exitmacro><#>
<endif>

<proc_call>(ConvertToUpper,"%vMyText%", "vMyText" )
<proc_call>(AddQuotes,"%vMyText%", "vMyText" )

<msg>(-100,-100,"%vMyText%","Result",1,0,0,0)
```

Procedure BEGIN: - < proc_def_begin >() ... [Pro]

Procedure BEGIN:
<proc_def_begin>(Identifier,Parameter)
Available in: Professional edition

This command defines begin of procedure. Procedure is a piece of macro code that can be called using "proc_call" command within the macro. Procedure can have any number of parameters. Procedures do not return a value but it is possible to pass results out of the procedure using reference parameters. Reference parameter contains name of the variable it references to. Enclose a parameter in between "&" characters to tell the procedure that the parameter is a reference (example: &parRefParam&).

Example: Let's have "proc_def_begin"(P1, "parParam1","&parRefParam&") and let's have procedure call "proc_call"(P1, "555", "vResult"). Now, any modification of parRefParam (that is is done within the P1 procedure) is actually done on vResult variable since the parRefParam references it.

Procedure parameter should starts with "par" prefix (for example, "parInputText"). This makes the parameter known (accessible) only locally within the procedure so that it does not conflict with other macro variables. Procedure local variables should start with "lpv" (Local Procedure Variable) prefix (for example, "lpvTemporaryVariable") for the same reason. It is not allowed to define procedure within other procedure (embedded "proc_def_begin"). The procedure definition must be ended by "proc_def_end" command.
When getting actual value of the referenced variable, it is necessary to enclose it to % (example: %parRefParam% provides actual value while just parRefParam provides name of the referenced variable).

#	Parameter name	Parameter description
1	Identifier	Unique name of the procedure. The name is used as a parameter in "proc_call" command to identify what procedure to call.
2	Parameter	Any number of parameters. Each parameter must be enclosed in between " character and parameters must be delimited by , Example: "parInputText","parTime","&parRefOutputText&"

Example (Macro Steps):

```
1   <#> <#> This example shows how to use procedures
2   Macro execution: ONLY COMMANDS
3   Procedure BEGIN: AddQuotes with parameters (parStringInput, &parStringOutput&)
4       Variable SET "parStringOutput="%parStringInput%"", Message text=""
5   Procedure END
6   Procedure BEGIN: ConvertToUpper with parameters (parStringInput,&parStringOutput&)
7       Variable OPERATION "STR_UPPER" (Variable for result = parStringOutput, Input text/variable =
        %parStringInput%, Parameter 1 = 2, Parameter 2 = , Parameter 3 = 0)
8   Procedure END
9   Variable SET "vMyText=", Message text="Insert text you want to convert to upper case and enclose to
    quotes:"
10  IF STRING _vCanceled==1
11      Macro EXIT
12  ENDIF
13  Procedure CALL: ConvertToUpper with parameters (%vMyText%, vMyText )
14  Procedure CALL: AddQuotes with parameters (%vMyText%, vMyText )
15  Message SHOW "Information" : "%vMyText%" (other parameters: x = -100, y = -100, Window title = Result,
    Buttons = OK, Timeout (seconds) = 0, Always on top = No).
```

Example (Plain Text):

```
<#> This example shows how to use procedures
<cmds>
<proc_def_begin>(AddQuotes,"parStringInput", "&parStringOutput&")
   <varset>("parStringOutput=%_vQuoteChar%%parStringInput%%_vQuoteChar%","")
<proc_def_end>

<proc_def_begin>(ConvertToUpper,"parStringInput","&parStringOutput&")
   <var_oper>(parStringOutput,"%parStringInput%",STR_UPPER,"2","", "0")
<proc_def_end>

<varset>("vMyText=","Insert text you want to convert to upper case and enclose to quotes:")
<if_str>("_vCanceled==1")<#>
<exitmacro><#>
<endif>

<proc_call>(ConvertToUpper,"%vMyText%", "vMyText" )
<proc_call>(AddQuotes,"%vMyText%", "vMyText" )

<msg>(-100,-100,"%vMyText%","Result",1,0,0,0)
```

Procedure CALL: - < proc_call >() ... [Pro]

Procedure CALL:
<proc_call>(Identifier,Parameter)
Available in: Professional edition

This command executes a procedure defined using "proc_def_begin" ... "proc_def_end" commands. When calling a procedure, it is always necessary to pass expected parameters (as they are defined in the "proc_def_begin" command). If a parameter is defined as a reference (see "proc_def_begin" help section for details) then it is necessary to pass an exisitng variable name as the parameter.

#	Parameter name	Parameter description
1	Identifier	Unique name of the procedure to call. It is expected that a procedure with this name is defined using command.
2	Parameter	Any number of parameters. Each parameter must be enclosed in between " character and parameters must be delimited by , Example: "%vInputText%","%vTime%","vOutputText"

Example (Macro Steps):

```
1   <#> <#> This example shows how to use procedures
2   Macro execution: ONLY COMMANDS
3   Procedure BEGIN: AddQuotes with parameters (parStringInput, &parStringOutput&)
4       Variable SET "parStringOutput="%parStringInput%"", Message text=""
5   Procedure END
6   Procedure BEGIN: ConvertToUpper with parameters (parStringInput,&parStringOutput&)
7       Variable OPERATION "STR_UPPER" (Variable for result = parStringOutput, Input text/variable =
        %parStringInput%, Parameter 1 = 2, Parameter 2 = , Parameter 3 = 0)
8   Procedure END
9   Variable SET "vMyText=", Message text="Insert text you want to convert to upper case and enclose to
    quotes:"
10  IF STRING _vCanceled==1
11      Macro EXIT
12  ENDIF
13  Procedure CALL: ConvertToUpper with parameters (%vMyText%, vMyText )
14  Procedure CALL: AddQuotes with parameters (%vMyText%, vMyText )
15  Message SHOW "Information" : "%vMyText%" (other parameters: x = -100, y = -100, Window title = Result,
    Buttons = OK, Timeout (seconds) = 0, Always on top = No).
```

Example (Plain Text):

```
<#> This example shows how to use procedures
<cmds>
<proc_def_begin>(AddQuotes,"parStringInput", "&parStringOutput&")
   <varset>("parStringOutput=%_vQuoteChar%%parStringInput%%_vQuoteChar%","")
<proc_def_end>

<proc_def_begin>(ConvertToUpper,"parStringInput","&parStringOutput&")
   <var_oper>(parStringOutput,"%parStringInput%",STR_UPPER,"2","", "0")
<proc_def_end>

<varset>("vMyText=","Insert text you want to convert to upper case and enclose to quotes:")
<if_str>("_vCanceled==1")<#>
<exitmacro><#>
<endif>

<proc_call>(ConvertToUpper,"%vMyText%", "vMyText" )
<proc_call>(AddQuotes,"%vMyText%", "vMyText" )

<msg>(-100,-100,"%vMyText%","Result",1,0,0,0)
```

INCLUDE here macro text from - < -include- >() ... [Pro]

INCLUDE here macro text from
<-include->("Identifier")
Available in: Professional edition

This command allows a user to include (insert) other macro or (text) file to the macro. Before macro execution is started, the <-include-> is replaced by content defined in the command (macro or file). This command makes it possible to use procedures (,) defined in external text (.mcr) files or in other macros.

#	Parameter name	Parameter description
1	Identifier	This parameter can be one of these: **macro:**Existing macro name (example: "macro:reusableProcedures") **file:**(Full) path to an existing file (example: "file:c:\.....\reusableProcedures.txt"

Example (Macro Steps):

1 <#> <#> This example shows how to use <-include-> command

2 **Macro execution: ONLY COMMANDS**

3 **INCLUDE here macro text from** "file:c:\MyMacroLibrary\lib.mcr"

4 **Procedure CALL:** VeryCommonTask with parameters (1,2,3)

Example (Plain Text):

```
<#> This example shows how to use <-include-> command
<cmds>
<-include->("file:c:\MyMacroLibrary\lib.mcr")
<proc_call>(VeryCommonTask,"1","2","3")
```

Procedure EXIT - < proc_exit > ... [Pro]

Procedure EXIT
<proc_exit>
Available in: Professional edition

This command exits procedure execution.

Example (Macro Steps):

```
1   <#> <#> This example shows how to use "proc_exit" command
2   Macro execution: ONLY COMMANDS
3   Procedure BEGIN: Example with parameters ()
4       Repeat steps UNTIL "i<1000" (Counter variable initial value = "i=0", Counter loop increment = "1")
5           IF i > 100
6               Message SHOW "Information" : "Exiting procedure execution" (other parameters: x = -100, y =
                -100, Window title = Result, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
7               Procedure EXIT
8           ENDIF
9       Repeat steps END
10  Procedure END
11  Procedure CALL: Example with parameters ()
12  Message SHOW "Information" : "Macro is finished" (other parameters: x = -100, y = -100, Window title =
    Result, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
```

Example (Plain Text):

```
<#> This example shows how to use "proc_exit" command
<cmds>
<proc_def_begin>(Example,)
   <for>("i=0","i<1000","1")
<if>("i > 100")
<msg>(-100,-100,"Exiting procedure execution","Result",1,0,0,0)
<proc_exit>
<endif>
<for_end>
<proc_def_end>

<proc_call>(Example,)

<msg>(-100,-100,"Macro is finished","Result",1,0,0,0)
```

Repeat steps UNTIL - < for >() ... [Pro]

Repeat steps UNTIL
<for>("Counter variable initial value","Condition","Counter loop increment")
Available in: Professional edition

This command begins loop that allows macro developer to repeat part of macro (between "for" and "for_end" commands) multiple times. Each "for" command must be followed by "for_end" command. Each loop, a condition defined in "for" command is evaluated and if it is evaluated as "not true" then the "for" loop is ended. It is possible to end the "for" loop also using "for_break" command.

#	Parameter name	Parameter description
1	Counter variable initial value	Counter variable initialization in form "Variable = value". Example: "varCounter = 1".
2	Condition	Condition in the same form as in "if_..." command. Example: "%varCounter% < 100".
3	Counter loop increment	The value that defines how to increase the counter value each loop. Example: "1".

Example (Macro Steps):

1 <#> <#> This macro shows how to use "for" loop.

2 <#> <#> There are two embedded "for" loops.

3 <#> <#>

4 **Macro execution: ONLY COMMANDS**

5 **Repeat steps UNTIL** "vX < 5" (Counter variable initial value = "vX=1", Counter loop increment = "1")

6 **Repeat steps UNTIL** "vY < 5" (Counter variable initial value = "vY=1", Counter loop increment = "1")

7 **Message SHOW** "Information" : "Counters [x,y] = [%vX%,%vY%] Press F10' to exit this macro." (other parameters: x = -100, y = -100, Window title = For Counters, Buttons = None, Timeout (seconds) = 0, Always on top =).

8 **Error message DISABLED**

9 **WAIT FOR** Object = "KEY", Event = "PRESS", Parameter = "", Timeout (seconds) = "1", Exact = "0"

10 **Error CLEAR**

11 **Error message ENABLED**

12 **If KEY / MOUSE BUTTON** "" is "Down" then execute following steps

13 **Macro EXIT**

14 **ENDIF**

15 **Repeat steps END**

16 **Repeat steps END**

Example (Plain Text):

```
<#> This macro shows how to use "for" loop.
<#> There are two embedded "for" loops.
<#>
<cmds>

<for>("vX=1","vX < 5","1")

<for>("vY=1","vY < 5","1")

<msg>(-100,-100,"Counters [x,y] = [%vX%,%vY%]

Press F10' to exit this macro.","For Counters",0,0,0)

<me_error_nodisplay> <waitfor>("KEY","PRESS","<F10>",1,0)<me_error_clear><me_error_display>

<if_key>("<F10>","DOWN")
<exitmacro>
<endif>

<for_end>

<for_end>
```

Repeat steps END - < for_end > ... [Pro]

Repeat steps END
<for_end>
Available in: Professional edition

This command ends the repeat section. See also "for" command.

Example (Macro Steps):

```
1   <#> <#> This macro shows how to use "for" loop.
2   <#> <#> There are two embedded "for" loops.
3   <#> <#>
4   Macro execution: ONLY COMMANDS
5   Repeat steps UNTIL "vX < 5" (Counter variable initial value = "vX=1", Counter loop increment = "1")
6       Repeat steps UNTIL "vY < 5" (Counter variable initial value = "vY=1", Counter loop increment = "1")
7           Message SHOW "Information" : "Counters [x,y] = [%vX%,%vY%] Press 'F10' to exit this macro." (other
            parameters: x = -100, y = -100, Window title = For Counters, Buttons = None, Timeout (seconds) = 0,
            Always on top = ).
8           Error message DISABLED
9           WAIT FOR Object = "KEY", Event = "PRESS", Parameter = "", Timeout (seconds) = "1", Exact = "0"
10          Error CLEAR
11          Error message ENABLED
12          If KEY / MOUSE BUTTON "" is "Down" then execute following steps
13              Macro EXIT
14          ENDIF
15      Repeat steps END
16  Repeat steps END
```

Example (Plain Text):

```
<#> This macro shows how to use "for" loop.
<#> There are two embedded "for" loops.
<#>
<cmds>

<for>("vX=1","vX < 5","1")
```

```
<for>("vY=1","vY < 5","1")

<msg>(-100,-100,"Counters [x,y] = [%vX%,%vY%]

Press 'F10' to exit this macro.","For Counters",0,0,0)

<me_error_nodisplay> <waitfor>("KEY","PRESS","<F10>",1,0)<me_error_clear><me_error_display>

<if_key>("<F10>","DOWN")
<exitmacro>
<endif>

<for_end>

<for_end>
```

Repeat steps BREAK - < for_break > ... [Pro]

Repeat steps BREAK
<for_break>
Available in: Professional edition

This command breaks a "for" loop execution and cause that the macro continues execution after "for_end" command of the broken "for" loop. See also "for" command.

Example (Macro Steps):

```
1   <#> <#> This macro shows how to use "for" loop.
2   <#> <#> There are two embedded "for" loops.
3   <#> <#>
4   Macro execution: ONLY COMMANDS
5   Repeat steps UNTIL "vX < 5" (Counter variable initial value = "vX=1", Counter loop increment = "1")
6       Repeat steps UNTIL "vY < 50" (Counter variable initial value = "vY=1", Counter loop increment = "1")
7           Message SHOW "Information" : "Counters [x,y] = [%vX%,%vY%] Press 'F10' to break the y-loop."
            (other parameters: x = -100, y = -100, Window title = For Counters, Buttons = None, Timeout (seconds)
            = 0, Always on top = ).
8           Error message DISABLED
9           WAIT FOR Object = "KEY", Event = "PRESS", Parameter = "", Timeout (seconds) = "1", Exact = "0"
10          Error CLEAR
11          Error message ENABLED
12          If KEY / MOUSE BUTTON "" is "Down" then execute following steps
13              Repeat steps BREAK
14          ENDIF
15      Repeat steps END
16  Repeat steps END
```

Example (Plain Text):

```
<#> This macro shows how to use "for" loop.
<#> There are two embedded "for" loops.
<#>
<cmds>

<for>("vX=1","vX < 5","1")
```

```
<for>("vY=1","vY < 50","1")

<msg>(-100,-100,"Counters [x,y] = [%vX%,%vY%]

Press 'F10' to break the y-loop.","For Counters",0,0,0)

<me_error_nodisplay> <waitfor>("KEY","PRESS","<F10>",1,0)<me_error_clear><me_error_display>

<if_key>("<F10>","DOWN")
   <for_break>
<endif>

<for_end>

<for_end>
```

IF WINDOWS SERVICE - < if_winsvc >() ... [Pro]

IF WINDOWS SERVICE
<if_winsvc>("Service Name","Condition")
Available in: Professional edition

The command evaluates a Windows service state and if it is evaluated as "true" then following macro steps (steps between "if" and "else/endif") are executed.

#	Parameter name	Parameter description
1	Service Name	The Windows service name
2	Condition	Windows service state: "Running" - the service is running "Not running" - the service is installed but it is not running. "Installed" - the service is installed, it does not matter if it is running or not. "Not Installed" - the service is not installed.

Example (Macro Steps):

```
1    <#> <#>This macro shows how to use "if_winsvc" and "winsvc" commands
2    Macro execution: ONLY COMMANDS
3    IF WINDOWS SERVICE "WebClient" is "NOT running" then execute following steps
4        WINDOWS SERVICE "WebClient", Command="Start"
5    ENDIF
```

Example (Plain Text):

```
<#>This macro shows how to use "if_winsvc" and "winsvc" commands
<cmds>
<if_winsvc>("WebClient","IS_NOT_RUNNING")
  <winsvc>("WebClient","START")
<endif>
```

Mouse Commands

MOVE - < mm >() ... [Free]

Mouse MOVE
<mm>(x,y,Time to move (ms))
Available in: Free edition

Moves mouse cursor to required position. The position is in absolute screen coordinates by default. There are these three commands that can precede the command and change the mouse move coordinates meaning: - mouse cursor position is in absolute screen coordinates - mouse cursor position is relative to currently active window - mouse cursor position is relative to current mouse cursor position

#	Parameter name	Parameter description
1	x	X-coordinate of the new mouse cursor position.
2	y	Y-coordinate of the new mouse cursor position.
3	Time to move (ms)	Time in milliseconds it is required the mouse move takes. Longer time means slower speed of the mouse. The time is adjusted by the macro playback speed settings.

Example (Macro Steps):

1 <#> <#> This macro moves the mouse cursor to position (100,100)

2 <#> <#> and than - after 2 seconds - to position (-10, -10) relative to

3 <#> <#> position (100, 100)

4 **Macro execution: ONLY COMMANDS**

5 **Mouse MOVE** position [x=100, y=100]

6 **WAIT** wait "2000" ms (time is constant: "")

7 **Mouse COORDINATES** are now RELATIVE to current MOUSE position

8 **Mouse MOVE** position [x=-10, y=-10]

Example (Plain Text):

```
<#> This macro moves the mouse cursor to position (100,100)
<#> and than - after 2 seconds - to position (-10, -10) relative to
<#> position (100, 100)
<cmds>

<mm>(100,100)

<wx>(2000)

<mousemove_relative_pos>
<mm>(-10,-10)
```

BUTTON: - < mlbd > ... [Free]

Mouse BUTTON:
<mlbd>
Available in: Free edition

Has the same effect as pressing left mouse button down.

Example (Macro Steps):

1	**<#> <#> This macro 'clicks' on position (100,100)**
2	**Macro execution: ONLY COMMANDS**
3	**Mouse MOVE** position [x=100, y=100]
4	**Mouse BUTTON:** LEFT button DOWN
5	**Mouse BUTTON:** LEFT button UP

Example (Plain Text):

```
<#> This macro 'clicks' on position (100,100)
<cmds>

<mm>(100,100)
<mlbd>
<mlbu>
```

BUTTON: - < mlbu > ... [Free]

Mouse BUTTON:
<mlbu>
Available in: Free edition

Has the same effect as releasing left mouse button (after it was pressed down).

Example (Macro Steps):

1	**<#> <#> This macro 'clicks' on position (100,100)**
2	**Macro execution: ONLY COMMANDS**
3	**Mouse MOVE** position [x=100, y=100]
4	**Mouse BUTTON:** LEFT button DOWN
5	**Mouse BUTTON:** LEFT button UP

Example (Plain Text):

```
<#> This macro 'clicks' on position (100,100)
<cmds>

<mm>(100,100)
<mlbd>
<mlbu>
```

BUTTON: - < mrbd > ... [Free]

Mouse BUTTON:
<mrbd>
Available in: Free edition

Has the same effect as pressing right mouse button down.

Example (Macro Steps):

1	**<#> <#> This macro 'right-clicks' on position (100,100)**
2	**Macro execution: ONLY COMMANDS**
3	**Mouse MOVE** position [x=100, y=100]
4	**Mouse BUTTON:** RIGHT button DOWN
5	**Mouse BUTTON:** RIGHT button UP

Example (Plain Text):

```
<#> This macro 'right-clicks' on position (100,100)
<cmds>

<mm>(100,100)
<mrbd>
<mrbu>
```

BUTTON: - < mrbu > ... [Free]

Mouse BUTTON:
<mrbu>
Available in: Free edition

Has the same effect as releasing right mouse button (after it was pressed down).

Example (Macro Steps):

1	**<#> <#> This macro 'right-clicks' on position (100,100)**
2	**Macro execution: ONLY COMMANDS**
3	**Mouse MOVE** position [x=100, y=100]
4	**Mouse BUTTON:** RIGHT button DOWN
5	**Mouse BUTTON:** RIGHT button UP

Example (Plain Text):

```
<#> This macro 'right-clicks' on position (100,100)
<cmds>

<mm>(100,100)
<mrbd>
<mrbu>
```

BUTTON: - < mmbd > ... [Free]

Mouse BUTTON:
<mmbd>
Available in: Free edition

Has the same effect as pressing middle mouse button down.

Example (Macro Steps):

1	**<#> <#> This macro 'middle-clicks' on position (100,100)**
2	**Macro execution: ONLY COMMANDS**
3	**Mouse MOVE** position [x=100, y=100]
4	**Mouse BUTTON:** MIDDLE button DOWN
5	**Mouse BUTTON:** MIDDLE button UP

Example (Plain Text):

```
<#> This macro 'middle-clicks' on position (100,100)
<cmds>

<mm>(100,100)
<mmbd>
<mmbu>
```

BUTTON: - < mmbu > ... [Free]

Mouse BUTTON:
<mmbu>
Available in: Free edition

Has the same effect as releasing middle mouse button (after it was pressed down).

Example (Macro Steps):

1	**<#> <#> This macro 'middle-clicks' on position (100,100)**
2	**Macro execution: ONLY COMMANDS**
3	**Mouse MOVE** position [x=100, y=100]
4	**Mouse BUTTON:** MIDDLE button DOWN
5	**Mouse BUTTON:** MIDDLE button UP

Example (Plain Text):

```
<#> This macro 'middle-clicks' on position (100,100)
<cmds>

<mm>(100,100)
<mmbd>
<mmbu>
```

COORDINATES - < mousemove_relative_win > ... [Free]

Mouse COORDINATES
<mousemove_relative_win>
Available in: Free edition

This command changes "mouse move" command behavior so that "mouse move" coordinates are relative to the upper-left corner of the active window.

Example (Macro Steps):

1 **<#> <#>This macro moves cursor to upper left corner of the active window**

2 **Mouse COORDINATES** are now RELATIVE to active WINDOW

3 **Mouse MOVE** position [x=0, y=0]

Example (Plain Text):

<#>This macro moves cursor to upper left corner of the active window
<mousemove_relative_win><mm>(0,0)

COORDINATES - < mousemove_absolute > ... [Free]

Mouse COORDINATES
<mousemove_absolute>
Available in: Free edition

Changes "mouse move" command behavior so that "mouse move" coordinates are in computer screen absolute coordinates (default behavior).

Example (Macro Steps):

1 **<#> <#>This macro moves cursor to position [100,100]**

2 **Mouse COORDINATES** are now ABSOLUTE

3 **Mouse MOVE** position [x=100, y=100]

Example (Plain Text):

<#>This macro moves cursor to position [100,100]
<mousemove_absolute><mm>(100,100)

COORDINATES - < mousemove_relative_pos > ... [Free]

Mouse COORDINATES
<mousemove_relative_pos>
Available in: Free edition

Changes "mouse move" command behavior so that "mouse move" coordinates are relative to current mouse position.

Example (Macro Steps):

1 <#> <#> This macro moves the mouse cursor to position (100,100)

2 <#> <#> and than - after 2 seconds - to position (-10, -10) relative to

3 <#> <#> position (100, 100)

4 **Macro execution: ONLY COMMANDS**

5 **Mouse MOVE** position [x=100, y=100]

6 **WAIT** wait "2000" ms (time is constant: "")

7 **Mouse COORDINATES** are now RELATIVE to current MOUSE position

8 **Mouse MOVE** position [x=-10, y=-10]

Example (Plain Text):

```
<#> This macro moves the mouse cursor to position (100,100)
<#> and than - after 2 seconds - to position (-10, -10) relative to
<#> position (100, 100)
<cmds>
<mm>(100,100)
<wx>(2000)
<mousemove_relative_pos>
<mm>(-10,-10)
```

COORDINATES - < mousemove_relative_definedwindow >() ... [Free]

Mouse COORDINATES
<mousemove_relative_definedwindow>("Window",Match)
Available in: Free edition

This command changes "mouse move" command behavior so that "mouse move" coordinates are relative to upper-left corner of the selected window.

#	Parameter name	Parameter description
1	Window	Window identifier in form of Window Identification Path (WIP) or HWND. It identifies the window that is to be closed.
2	Match	Takes effect only if a window is identified using WIP parameter. Can be one of these values: 0 - match substrings in WIP 1 - match exact strings in WIP

Example (Macro Steps):

```
1   <#> <#>This example moves mouse cursor to upper-left corner of Notepad window
2   Macro execution: ONLY COMMANDS
3   IF WINDOW "[* - Notepad|Notepad|#0|#114]" Is Open (Match=Partial)
4       Mouse COORDINATES are now RELATIVE to WINDOW "[* - Notepad|Notepad|#0|#114]" (Match = Partial)
5       Mouse MOVE position [ x=10, y=10 ]
6   ENDIF
```

Example (Plain Text):

```
<#>This example moves mouse cursor to upper-left corner of Notepad window
<cmds>
<if_win>("[* - Notepad|Notepad|#0|#114]","OPEN",0)
  <mousemove_relative_definedwindow>("[* - Notepad|Notepad|#0|#114]",0)
  <mm>(10,10)
<endif>
```

BUTTON: - < mx1bd > ... [Free]

Mouse BUTTON:
<mx1bd>
Available in: Free edition

Has the same effect as pressing "X1" mouse button down.

Example (Macro Steps):

1	**<#> <#> This macro does X1 click on position (100,100)**
2	**Macro execution: ONLY COMMANDS**
3	**Mouse MOVE** position [x=100, y=100]
4	**Mouse BUTTON:** X1 button DOWN
5	**Mouse BUTTON:** X1 button UP

Example (Plain Text):

```
<#> This macro does X1 click on position (100,100)
<cmds>

<mm>(100,100)
<mx1bd>
<mx1bu>
```

BUTTON: - < mx1bu > ... [Free]

Mouse BUTTON:
<mx1bu>
Available in: Free edition

Has the same effect as releasing "X1" mouse button (after it was pressed down).

Example (Macro Steps):

1	**<#> <#> This macro does X1 click on position (100,100)**
2	**Macro execution: ONLY COMMANDS**
3	**Mouse MOVE** position [x=100, y=100]
4	**Mouse BUTTON:** X1 button DOWN
5	**Mouse BUTTON:** X1 button UP

Example (Plain Text):

```
<#> This macro does X1 click on position (100,100)
<cmds>

<mm>(100,100)
<mx1bd>
<mx1bu>
```

BUTTON: - < mx2bd > ... [Free]

Mouse BUTTON:
<mx2bd>
Available in: Free edition

Has the same effect as pressing "X2" mouse button down.

Example (Macro Steps):

1	**<#> <#> This macro does X2 click on position (100,100)**
2	**Macro execution: ONLY COMMANDS**
3	**Mouse MOVE** position [x=100, y=100]
4	**Mouse BUTTON:** X2 button DOWN
5	**Mouse BUTTON:** X2 button UP

Example (Plain Text):

```
<#> This macro does X2 click on position (100,100)
<cmds>

<mm>(100,100)
<mx2bd>
<mx2bu>
```

BUTTON: - < mx2bu > ... [Free]

Mouse BUTTON:
<mx2bu>
Available in: Free edition

Has the same effect as releasing "X2" mouse button (after it was pressed down).

Example (Macro Steps):

1	**<#> <#> This macro does X2 click on position (100,100)**
2	**Macro execution: ONLY COMMANDS**
3	**Mouse MOVE** position [x=100, y=100]
4	**Mouse BUTTON:** X2 button DOWN
5	**Mouse BUTTON:** X2 button UP

Example (Plain Text):

```
<#> This macro does X2 click on position (100,100)
<cmds>

<mm>(100,100)
<mx2bd>
<mx2bu>
```

BLOCK - < mouse_block > ... [Pro]

Mouse BLOCK
<mouse_block>
Available in: Professional edition

This command blocks mouse events. It can be used when it is necessary to disable mouse events during macro execution, for example, before "wait for mouse" command ("waitfor"). If it is required to disable keyboard and mouse input during whole macro execution then it is also possible to use "Lock keyboard and mouse while macro is running" option in the macro settings tab. To unblock mouse, use "mouse_unblock" command.

Example (Macro Steps):

1 <#> <#> This macro shows how to use 'mouse_block' and 'mouse_unblock' commands

2 **Macro execution: ONLY COMMANDS**

3 **Message SHOW** "Information" : "Waiting untill right-click ...." (other parameters: x = 32, y = 32, Window title = Message, Buttons = None, Timeout (seconds) = 0, Always on top = No).

4 **Mouse BLOCK**

5 **WAIT FOR** Object = "MOUSE", Event = "PRESS", Parameter = "", Timeout (seconds) = "10", Exact = "0"

6 **Mouse UNBLOCK**

7 **Message CLOSE**

Example (Plain Text):

```
<#> This macro shows how to use 'mouse_block' and 'mouse_unblock' commands
<cmds>
<msg>(32,32,"Waiting untill right-click ....","Message",0,0,0,0)

<mouse_block>
<waitfor>("MOUSE","PRESS","<mrbd>",10,0)

<mouse_unblock>
<msgoff>
```

UNBLOCK - < mouse_unblock > ... [Pro]

Mouse UNBLOCK
<mouse_unblock>
Available in: Professional edition

This command unblocks mouse events blocked by previous use of "mouse_block" command.

Example (Macro Steps):

1 **<#> <#> This macro shows how to use 'mouse_block' and 'mouse_unblock' commands**

2 **Macro execution: ONLY COMMANDS**

3 **Message SHOW** "Information" : "Waiting untill right-click ...." (other parameters: x = 32, y = 32, Window title = Message, Buttons = None, Timeout (seconds) = 0, Always on top = No).

4 **Mouse BLOCK**

5 **WAIT FOR** Object = "MOUSE", Event = "PRESS", Parameter = "", Timeout (seconds) = "10", Exact = "0"

6 **Mouse UNBLOCK**

7 **Message CLOSE**

Example (Plain Text):

```
<#> This macro shows how to use 'mouse_block' and 'mouse_unblock' commands
<cmds>
<msg>(32,32,"Waiting untill right-click ....","Message",0,0,0,0)

<mouse_block>
<waitfor>("MOUSE","PRESS","<mrbd>",10,0)

<mouse_unblock>
<msgoff>
```

WHEEL FORWARD - < mwheel_f > ... [Free]

Mouse WHEEL FORWARD
<mwheel_f>
Available in: Free edition

This command has the same effect as rotating mouse wheel forward.

Example (Macro Steps):

1	**<#> <#> This macro scrolls a little bit using mouse wheel**
2	**Mouse WHEEL FORWARD**

Example (Plain Text):

```
<#> This macro scrolls a little bit using mouse wheel
<mwheel_f>
```

WHEEL BACKWARD - < mwheel_b > ... [Free]

Mouse WHEEL BACKWARD
<mwheel_b>
Available in: Free edition

This command has the same effect as rotating mouse wheel backward.

Example (Macro Steps):

1 **<#> <#> This macro scrolls a little bit using mouse wheel**

2 **Mouse WHEEL BACKWARD**

Example (Plain Text):

<#> This macro scrolls a little bit using mouse wheel
<mwheel_b>

DOUBLE-CLICK - < m2click > ... [Free]

Mouse DOUBLE-CLICK
<m2click>
Available in: Free edition

This command has the same effect as left mouse button double-click.

Example (Macro Steps):

1	**<#> <#> This macro 'double-clicks' on position (100,100)**
2	**Mouse MOVE** position [x=100, y=100]
3	**Mouse DOUBLE-CLICK**

Example (Plain Text):

```
<#> This macro 'double-clicks' on position (100,100)
<mm>(100,100)<m2click>
```

Networking/Web/E-mail

Web OPEN PAGE - < wwwopen >() ... [Pro]

Web OPEN PAGE
<wwwopen>("URL",Window state,Time to wait,Window handle,Browser)
Available in: Professional edition

Opens web page in a web browser.
Note: In order to use this command together with "www_fillform" command the "Internet Explorer" browser must be used.

#	Parameter name	Parameter description
1	URL	Link (URL) to the web page (e.g., "http://www.macrotoolworks.com"). The link must contain http://, https://, etc. prefix.
2	Window state	The state of the window: 0 - Normal 1 - Maximized 2 - Minimized
3	Time to wait	Time in seconds. If the web page is not opened in this time frame then the command fails.
4	Window handle	The value of this field can be one of these: • Variable name - variable containing handle identifying web browser window to use. If the variable value is "empty" then a new web browser window is opened. The variable receives handle of this new browser window. • 0 or Empty - if the field is empty or contains 0 then an already opened window is used. • 1 - if the field contains 1 then a new Internet Explorer window is open. • 2 - if the field contains 2 then a new Internet Explorer tab is open.
5	Browser	Browser to use: IE - Internet Explorer Default - The browser that is set as default is used. In this case the previous "Time to wait" parameter and "Window handle" parameters are ignored.

Example (Macro Steps):

1 <#> <#> This command opens pitrinec.com web page

2 **Macro execution: ONLY COMMANDS**

3 **Web OPEN PAGE** URL = http://wwww.pitrinec.com, Window state = Normal, Time to wait = 15, Window handle = , Browser = Internet Explorer

Example (Plain Text):

```
<#> This command opens pitrinec.com web page
<#>
<cmds>
<wwwopen>("http://wwww.pitrinec.com",0,15,,IE)
```

Net drive CONNECT - < netcondrive >() ... [Pro]

Net drive CONNECT
<netcondrive>("Drive letter","Network path","Login name","Password","Restore at next logon")
Available in: Professional edition

Maps (connects) a network folder to local drive.

#	Parameter name	Parameter description
1	Drive letter	Drive you want to map the network folder to (e.g., "X:"). It must be one letter followed by : .
2	Network path	Network folder (for example, "\\server\mydir").
3	Login name	User name to use to gain access rights to the network folder.
4	Password	Password to use to gain access rights to the network folder.
5	Restore at next logon	Restore at next logon: 0 - the drive is not mapped at next logon 1 - the drive will be automatically mapped at next logon

Example (Macro Steps):

1 <#> <#> This macro maps a network directory to local drive.

2 Macro execution: ONLY COMMANDS

3 Net drive CONNECT "X:" = "\\server\mzdir" (Login name=john, Restore at next logon=No)

Example (Plain Text):

```
<#> This macro maps a network directory to local drive.
<#>
<cmds>
<netcondrive>("X:","\\server\mzdir","john","jpwd",0)
```

Net drive DISCONNECT - < netdiscondrive >() ... [Pro]

Net drive DISCONNECT
<netdiscondrive>("Drive letter",Restore at next logon)
Available in: Professional edition

Disconnects a network folder from a local drive.

#	Parameter name	Parameter description
1	Drive letter	Drive you want to remove (e.g., "X:"). It must be one letter followed by : .
2	Restore at next logon	Restore at next logon: 0 - the drive will not be automatically mapped at next logon 1 - the drive will be automatically mapped at next logon

Example (Macro Steps):

1 <#> <#> This macro disconnects drive from a network folder

2 Macro execution: ONLY COMMANDS

3 Net drive DISCONNECT "X:", Restore at next logon=" 0"

Example (Plain Text):

```
<#> This macro disconnects drive from a network folder
<#>
<cmds>
<netdiscondrive>("X:", 0)
```

ftp GET - < ftp_getfile >() ... [Pro]

ftp GET
<ftp_getfile>("File","Remote file","Login name","Password","Silent")
Available in: Professional edition

Downloads file from a remote FTP server.

#	Parameter name	Parameter description
1	File	Full path to the file on local machine. If the file already exists it will be overwritten without a prior prompt.
2	Remote file	Full path to the remote file (e.g., ftp://pitrinec.com/pub/test.txt). The file path can contain * and ? wildcards. In such a case the "LocalFile" parameter must contain folder where to store multiple files received.
3	Login name	User login name. If empty the "anonymous" is considered.
4	Password	User name login password.
5	Silent	Silent mode. If set to 1 then the operation progress window is not displayed.

Example (Macro Steps):

```
1   <#> <#> This macro downloads "test.txt" file from pitrinec.com
2   <#> <#>
3   Macro execution: ONLY COMMANDS
4   Message SHOW "" : "Where To Save Downloaded File?" (other parameters: x = 32, y = 32, Window title = Message, Buttons = None, Timeout (seconds) = , Always on top = ).
5   Variable OPERATION "SELECT_FOLDER" (Variable for result = vLocalFolder, Input text/variable = , Parameter 1 = , Parameter 2 = , Parameter 3 = 0)
6   IF STRING _vCanceled==1
7       Macro EXIT
8   ENDIF
9   Message CLOSE
10  ftp GET "ftp://pitrinec.com/pub/test.txt" (Login name=anonymous) to file "%vLocalFolder%\test.txt"
11  Message SHOW "" : "File is downloaded. Do you want to open it?" (other parameters: x = -100, y = -100, Window title = Message, Buttons = Yes and No, Timeout (seconds) = , Always on top = ).
12  IF STRING _vMsgButton==YES
13      File OPEN open file "%vLocalFolder%\test.txt" in system default viewer.
14  ENDIF
```

Example (Plain Text):

```
<#> This macro downloads "test.txt" file from pitrinec.com
<#>
<cmds>

<msg>(32,32,"Where To Save Downloaded File?","Message",0)
<var_oper>(vLocalFolder,"",SELECT_FOLDER,"","", "0")
<if_str>("_vCanceled==1")<#>
<exitmacro><#>
<endif>
<msgoff>

<ftp_getfile>("%vLocalFolder%\test.txt","ftp://pitrinec.com/pub/test.txt","anonymous","")

<msg>(-100,-100,"File is downloaded. Do you want to open it?","Message",2)
<if_str>("_vMsgButton==YES")
   <fileopen>("%vLocalFolder%\test.txt",0)
<endif>
```

ftp PUT - < ftp_putfile >() ... [Pro]

ftp PUT
<ftp_putfile>("File","Remote file","Login name","Password","Silent")
Available in: Professional edition

Uploads file(s) to a remote FTP server.

#	Parameter name	Parameter description
1	File	Path to the file on local machine. The path can contain * and ? wildcards.
2	Remote file	Path to the remote file (e.g., ftp://pitrinec.com/pub/about.txt). If the "LocalFile" field contains wildcards then this field specifies directory where to upload multiple files (e.g., ftp://pitrinec.com/pub/).
3	Login name	User login name. If empty the "anonymous" is considered.
4	Password	User name login password.
5	Silent	Silent mode. If set to 1 then the operation progress window is not displayed.

Example (Macro Steps):

1 <#> <#> This macro uploads files to pitrinec.com

2 <#> <#>

3 **ftp PUT** upload "c:\temp_c*.txt" to Ftp site "ftp://pitrinec.com/anon_ftp/pub/" (Login name=YOUR_USER_NAME)

Example (Plain Text):

```
<#> This macro uploads files to pitrinec.com
<#>
<ftp_putfile>("c:\temp\_c*.txt","ftp://pitrinec.com/anon_ftp/pub/","YOUR_USER_NAME","YOUR_PASSWORD")
```

ftp DELETE - < ftp_delfile >() ... [Pro]

ftp DELETE
<ftp_delfile>("Unused","Remote file","Login name","Password","Silent")
Available in: Professional edition

Deletes file from a remote FTP server.

#	Parameter name	Parameter description
1	Unused	Must be empty.
2	Remote file	Full path to the remote file (e.g., ftp://softwareutilities.com/about.txt). The path can contain * and ? wildcards.
3	Login name	User login name. If empty the "anonymous" is considered.
4	Password	User name login password.
5	Silent	Silent mode. If set to 1 then the operation progress window is not displayed.

Example (Macro Steps):

```
1   <#> <#> This macro deletes file from FTP server
2   Macro execution: ONLY COMMANDS
3   Form FIELD "Remote file to delete:" of type "Text edit" (Default value=, Variable to save field value=vRemoteFile, Form identifier=fm1)
4   Form FIELD "User:" of type "Text edit" (Default value=, Variable to save field value=vUser, Form identifier=fm1)
5   Form FIELD "Password:" of type "Text edit" (Default value=, Variable to save field value=vPassword, Form identifier=fm1)
6   Form OPEN "fm1", Window title="FTP: Delete File"
7   IF STRING _vCanceled==1
8       Macro EXIT
9   ENDIF
10  ftp DELETE "vRemoteFile" (Login name=vUser)
```

Example (Plain Text):

```
<#> This macro deletes file from FTP server
<#>
<cmds>

<form_item>("fm1","Remote file to delete:","EDIT","","vRemoteFile")
<form_item>("fm1","User:","EDIT","","vUser")
<form_item>("fm1","Password:","EDIT","","vPassword")

<form_show>("fm1","FTP: Delete File","shell32.dll",32)
<if_str>("_vCanceled==1")<exitmacro><endif>
```

<ftp_delfile>("","vRemoteFile","vUser","vPassword")

ftp RENAME FILE - < ftp_renamefile >() ... [Pro]

ftp RENAME FILE
<ftp_renamefile>("Old name","New name","Login name","Password","Silent")
Available in: Professional edition

Renames file on a remote FTP server.

#	Parameter name	Parameter description
1	Old name	Full path to the original remote file (e.g., ftp://softwareutilities.com/original.txt).
2	New name	Full path to the remote new file (e.g., ftp://softwareutilities.com/renamed.txt).
3	Login name	User login name. If empty the "anonymous" is considered.
4	Password	User name login password.
5	Silent	Silent mode. If set to 1 then the operation progress window is not displayed.

Example (Macro Steps):

```
1   <#> <#> This macro uploads file to an FTP server
2   Macro execution: ONLY COMMANDS
3   Form FIELD "Remote file - old name:" of type "Text edit" (Default value=, Variable to save field value=vRemoteFileOld, Form identifier=fm1)
4   Form FIELD "Remote file - new name:" of type "Text edit" (Default value=, Variable to save field value=vRemoteFileNew, Form identifier=fm1)
5   Form FIELD "User:" of type "Text edit" (Default value=, Variable to save field value=vUser, Form identifier=fm1)
6   Form FIELD "Password:" of type "Text edit" (Default value=, Variable to save field value=vPassword, Form identifier=fm1)
7   Form OPEN "fm1", Window title="File Rename"
8   IF STRING _vCanceled==1
9       Macro EXIT
10  ENDIF
11  ftp RENAME FILE "vRemoteFileOld" to "vRemoteFileNew" (Login name=vUser)
```

Example (Plain Text):

```
<#> This macro uploads file to an FTP server
<#>
<cmds>

<form_item>("fm1","Remote file - old name:","EDIT","","vRemoteFileOld")
<form_item>("fm1","Remote file - new name:","EDIT","","vRemoteFileNew")
<form_item>("fm1","User:","EDIT","","vUser")
<form_item>("fm1","Password:","EDIT","","vPassword")
```

```
<form_show>("fm1","File Rename","shell32.dll",1)
<if_str>("_vCanceled==1")<exitmacro><endif>

<ftp_renamefile>("vRemoteFileOld","vRemoteFileNew","vUser","vPassword")
```

ftp CREATE DIRECTORY - < ftp_createdir >() ... [Pro]

ftp CREATE DIRECTORY
<ftp_createdir>("Unused","New name","Login name","Password","Silent")
Available in: Professional edition

Creates new directory on a remote FTP server.

#	Parameter name	Parameter description
1	Unused	Must be empty.
2	New name	Full path to the remote directory to create (e.g., ftp://softwareutilities.com/newdir).
3	Login name	User login name. If empty the "anonymous" is considered.
4	Password	User name login password.
5	Silent	Silent mode. If set to 1 then the operation progress window is not displayed.

Example (Macro Steps):

```
1   <#> <#> This macro creates new directory on FTP server
2   Macro execution: ONLY COMMANDS
3   Form FIELD "Remote directory to create:" of type "Text edit" (Default value=, Variable to save field value=vRemoteDir, Form identifier=fm1)
4   Form FIELD "User:" of type "Text edit" (Default value=, Variable to save field value=vUser, Form identifier=fm1)
5   Form FIELD "Password:" of type "Text edit" (Default value=, Variable to save field value=vPassword, Form identifier=fm1)
6   Form OPEN "fm1", Window title="FTP: Create Directory"
7   IF STRING _vCanceled==1
8       Macro EXIT
9   ENDIF
10  ftp CREATE DIRECTORY "vRemoteDir" (Login name=vUser)
```

Example (Plain Text):

```
<#> This macro creates new directory on FTP server
<#>
<cmds>

<form_item>("fm1","Remote directory to create:","EDIT","","vRemoteDir")
<form_item>("fm1","User:","EDIT","","vUser")
<form_item>("fm1","Password:","EDIT","","vPassword")

<form_show>("fm1","FTP: Create Directory","shell32.dll",3)
<if_str>("_vCanceled==1")<exitmacro><endif>
```

<ftp_createdir>("","vRemoteDir","vUser","vPassword")

ftp DELETE DIRECTORY - < ftp_deldir >() ... [Pro]

ftp DELETE DIRECTORY
<ftp_deldir>("Unused","Delete","Login name","Password","Silent")
Available in: Professional edition

Deletes directory on a remote FTP server.

#	Parameter name	Parameter description
1	Unused	Must be empty.
2	Delete	Full path to the remote directory to delete (e.g., ftp://softwareutilities.com/dir).
3	Login name	User login name. If empty the "anonymous" is considered.
4	Password	User name login password.
5	Silent	Silent mode. If set to 1 then the operation progress window is not displayed.

Example (Macro Steps):

```
1   <#> <#> This macro deletes directory on FTP server
2   Macro execution: ONLY COMMANDS
3   Form FIELD "Remote directory to delete:" of type "Text edit" (Default value=, Variable to save field value=vRemoteDir, Form identifier=fm1)
4   Form FIELD "User:" of type "Text edit" (Default value=, Variable to save field value=vUser, Form identifier=fm1)
5   Form FIELD "Password:" of type "Text edit" (Default value=, Variable to save field value=vPassword, Form identifier=fm1)
6   Form OPEN "fm1", Window title="FTP: Delete Directory"
7   IF STRING _vCanceled==1
8       Macro EXIT
9   ENDIF
10  ftp DELETE DIRECTORY "vRemoteDir" (Login name=vUser)
```

Example (Plain Text):

```
<#> This macro deletes directory on FTP server
<#>
<cmds>

<form_item>("fm1","Remote directory to delete:","EDIT","","vRemoteDir")
<form_item>("fm1","User:","EDIT","","vUser")
<form_item>("fm1","Password:","EDIT","","vPassword")

<form_show>("fm1","FTP: Delete Directory","shell32.dll",31)
<if_str>("_vCanceled==1")<exitmacro><endif>
```

<ftp_deldir>("","vRemoteDir","vUser","vPassword")

E-mail SEND - < email_send >() ... [Pro]

E-mail SEND
<email_send>("Subject","Message text","To","CC","BCC","Attachment")
Available in: Professional edition

This command sends an e-mail without any user's interaction using default e-mail client.

#	Parameter name	Parameter description
1	Subject	E-mail subject field.
2	Message text	Message text.
3	To	E-mail address where to send the e-mail. Example: someone@somewhere.net
4	CC	E-mail address of whom to send CC (carbon copy). Example: someone@somewhere.net
5	BCC	E-mail address of whom to send BCC (blind carbon copy). Example: someone@somewhere.net
6	Attachment	Full path to files that will be sent along with the e-mail message. The files are delimited by comma.

Example (Macro Steps):

1 <#> <#> This command automatically sends an e-mail message

2 **Macro execution: ONLY COMMANDS**

3 **E-mail SEND** Subject=Hello, Message text=Hello, this is just a sample e-mail..., To=someone@somewhere.net, CC = , BCC = , Attachment = C:\documents\doc1.doc

Example (Plain Text):

```
<#> This command automatically sends an e-mail message
<#>
<cmds>
<email_send>("Hello","Hello,
this is just a sample e-mail...","someone@somewhere.net","","","C:\documents\doc1.doc")
```

Http DOWNLOAD - < download >() ... [Pro]

Http DOWNLOAD
<download>("File","Remote file","Login name","Password")
Available in: Professional edition

Downloads file from a HTTP server.

#	Parameter name	Parameter description
1	File	(Full) path to the file on local machine. If the file already exists it will be overwritten without a prior prompt.
2	Remote file	Full path to the remote file (e.g., http://www.pitrinec.com/index.html).
3	Login name	User login name. If empty the "anonymous" is considered.
4	Password	User name login password.

Example (Macro Steps):

```
1   <#> <#> This sample macro downloads home page file from www.pitrinec.com
2   <#> <#>
3   Macro execution: ONLY COMMANDS
4   Message SHOW "" : "Where To Save Downloaded File?" (other parameters: x = 32, y = 32, Window title = Message, Buttons = None, Timeout (seconds) = , Always on top = ).
5   Variable OPERATION "SELECT_FOLDER" (Variable for result = vLocalFolder, Input text/variable = , Parameter 1 = , Parameter 2 = , Parameter 3 = 0)
6   IF STRING _vCanceled==1
7       Macro EXIT
8   ENDIF
9   Message CLOSE
10  Http DOWNLOAD "http://www.pitrinec.com/index.html" (Login name=) to file "%vLocalFolder%\index.html"
11  Message SHOW "" : "File is downloaded. Do you want to open it?" (other parameters: x = -100, y = -100, Window title = Message, Buttons = Yes and No, Timeout (seconds) = , Always on top = ).
12  IF STRING _vMsgButton==YES
13      File OPEN open file "%vLocalFolder%\index.html" in system default viewer.
14  ENDIF
```

Example (Plain Text):

```
<#> This sample macro downloads home page file from www.pitrinec.com
<#>
```

```
<cmds>

<msg>(32,32,"Where To Save Downloaded File?","Message",0)
<var_oper>(vLocalFolder,"",SELECT_FOLDER,"","", "0")
<if_str>("_vCanceled==1")<exitmacro><endif>
<msgoff>

<download>("%vLocalFolder%\index.html","http://www.pitrinec.com/index.html","","")

<msg>(-100,-100,"File is downloaded. Do you want to open it?","Message",2)
<if_str>("_vMsgButton==YES")
   <fileopen>("%vLocalFolder%\index.html",0)
<endif>
```

ftp GET FILE SIZE - < ftp_filesize >() ... [Pro]

ftp GET FILE SIZE
<ftp_filesize>("Variable for result","Remote file","Login name","Password","Silent")
Available in: Professional edition

This command retrieves size of a file on a remote FTP server.

#	Parameter name	Parameter description
1	Variable for result	Variable that receives file size in bytes.
2	Remote file	Full path to the remote file (e.g., ftp://pitrinec.com/pub/test.txt).
3	Login name	User login name. If empty the "anonymous" is considered.
4	Password	User name login password.
5	Silent	Silent mode. If set to 1 then the operation progress window is not displayed.

Example (Macro Steps):

1 <#> <#> This sample retrieves size of file placed on remote FTP server

2 <#> <#>

3 **Macro execution: ONLY COMMANDS**

4 **ftp GET FILE SIZE** Variable for result=vFileSize, Remote file=ftp://pitrinec.com/pub/test.txt, Login name=

5 **Message SHOW** "" : "Size of "ftp://pitrinec.com/pub/test.txt" file is %vFileSize% bytes." (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

Example (Plain Text):

```
<#> This sample retrieves size of file placed on remote FTP server
<#>
<cmds>
<ftp_filesize>("vFileSize","ftp://pitrinec.com/pub/test.txt","","")
<msg>(-100,-100,"Size of %_vQuoteChar%ftp://pitrinec.com/pub/test.txt%_vQuoteChar% file is %vFileSize%
bytes.","Message",1)
```

ftp GET FILE MODIFICATION TIME - < ftp_filetime >() ... [Pro]

ftp GET FILE MODIFICATION TIME
<ftp_filetime>("Variable for result","Remote file","Login name","Password","Silent")
Available in: Professional edition

This command retrieves last modification time of a file on a remote FTP server. The time is in form: "MM/DD/YYYY Hour:Minute:Second".

#	Parameter name	Parameter description
1	Variable for result	Variable that receives last file modification time in "MM/DD/YYYY Hour:Minute:Second" form.
2	Remote file	Full path to the remote file (e.g., ftp://pitrinec.com/pub/test.txt).
3	Login name	User login name. If empty the "anonymous" is considered.
4	Password	User name login password.
5	Silent	Silent mode. If set to 1 then the operation progress window is not displayed.

Example (Macro Steps):

```
1   <#> <#> This sample retrieves last modification time of file placed on remote FTP server
2   <#> <#>
3   Macro execution: ONLY COMMANDS
4   ftp GET FILE MODIFICATION TIME Variable for result=vFileTime, Remote file=ftp://pitrinec.com/pub/test.txt, Login name=
5   Message SHOW "" : "Last modification time of "ftp://pitrinec.com/pub/test.txt" file is %vFileTime%." (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
6   Message SHOW "" : "Do you want to parse the time?" (other parameters: x = -100, y = -100, Window title = Message, Buttons = Yes and No, Timeout (seconds) = , Always on top = ).
7   IF STRING _vMsgButton==YES
8       Variable PARSE "vFileTime" to variable array "vTimeItems" (Delimiter = / :, Trim character = , Variable array for enumerated items = vTimeItems, Variable array size = vNumberOfTimeItems)
9       Loop BEGIN Repeat = vNumberOfTimeItems
10          Message SHOW "" : "%vTimeItems[_vLoopCounter0]%" (other parameters: x = -100, y = -100, Window title = Time items:, Buttons = OK, Timeout (seconds) = , Always on top = ).
11      Loop END
12  ENDIF
```

Example (Plain Text):

```
<#> This sample retrieves last modification time of file placed on remote FTP server
```

```
<#>
<cmds>

<ftp_filetime>("vFileTime","ftp://pitrinec.com/pub/test.txt","","")
<msg>(-100,-100,"Last modification time of %_vQuoteChar%ftp://pitrinec.com/pub/test.txt%_vQuoteChar% file is
%vFileTime%.","Message",1)

<msg>(-100,-100,"Do you want to parse the time?","Message",2)
<if_str>("_vMsgButton==YES")
  <var_parse>("vFileTime","/ :","",vTimeItems,vNumberOfTimeItems)
  <begloop>(vNumberOfTimeItems)
    <msg>(-100,-100,"%vTimeItems[_vLoopCounter0]%","Time items:",1)
  <endloop>
<endif>
```

E-mail POP3: GET LIST - < email_pop3_getlist >() ... [Pro]

E-mail POP3: GET LIST
<email_pop3_getlist>("Connection Id","Variable receiving number of e-mails","Variable array with e-mail senders","Variable array with e-mail subjects")
Available in: Professional edition

This command retrieves list of messages waiting on POP3 e-mail account. Before command can be called a connection to the e-mail account needs to be opened using command.

#	Parameter name	Parameter description
1	Connection Id	An identifier of connection to the e-mail account. Connection is opened using command.
2	Variable receiving number of e-mails	Variable that receives number of e-mail messages waiting on e-mail account.
3	Variable array with e-mail senders	Variable that receives number of e-mail messages waiting on e-mail account.
4	Variable array with e-mail subjects	Variable (array) that receives "subject" field of a waiting e-mail message.

Example (Macro Steps):

```
1   <#> <#> Thic sample shows how to get list of e-mails received on POP3 account.
2   Macro execution: ONLY COMMANDS
3   <#> <#> Connect to POP3 account first:
4   E-mail POP3: CONNECT Server=myemails.com, Login name=myaccount, Password=737qw7523#$,
    Connection Id = POP3Connection, Port = , TLS/SSL =
5   <#> <#> Get list of messages then:
6   E-mail POP3: GET LIST Connection Id = "POP3Connection", Variable receiving number of e-mails =
    "vNumberOfMessages", Variable array with e-mail senders = "vMessages_From", Variable array with e-mail
    subjects = "vMessages_Subject"
7   <#> <#> Disconnect from the account:
8   E-mail POP3: DISCONNECT Connection Id = POP3Connection
9   <#> <#> Show the list of messages:
10  IF NUMERIC vNumberOfMessages > 0
11      Variable SET "vEmailsList=_vStrEmpty", Message text=""
12      Repeat steps UNTIL "vEmails < vNumberOfMessages" (Counter variable initial value = "vEmails=0",
        Counter loop increment = "1")
13          Variable OPERATION "STR_APPEND" (Variable for result = vEmailsList, Input text/variable =
            %vEmailsList%, Parameter 1 = vMessages_Subject[vEmails], Parameter 2 = , Parameter 3 = 0)
14          Variable OPERATION "STR_APPEND" (Variable for result = vEmailsList, Input text/variable =
            %vEmailsList%, Parameter 1 = %_vKeyNewLine%, Parameter 2 = , Parameter 3 = 0)
15      Repeat steps END
16      Message SHOW "Information" : "%vEmailsList%" (other parameters: x = -100, y = -100, Window title =
        Emails received, Buttons = OK, Timeout (seconds) = 0, Always on top = ).
17  ELSE activate
18      Message SHOW "Error" : "There are no e-mails received." (other parameters: x = -100, y = -100, Window
        title = Emails received, Buttons = OK, Timeout (seconds) = 0, Always on top = ).
19  ENDIF
```

Example (Plain Text):

```
<#> Thic sample shows how to get list of e-mails received on POP3 account.
<cmds>

<#> Connect to POP3 account first:
<email_pop3_connect>("myemails.com","myaccount","737qw7523#$","POP3Connection")

<#> Get list of messages then:
<email_pop3_getlist>("POP3Connection","vNumberOfMessages","vMessages_From","vMessages_Subject")

<#> Disconnect from the account:
<email_pop3_disconnect>("POP3Connection")
```

```
<#> Show the list of messages:
<if_num>("vNumberOfMessages > 0")

   <varset>("vEmailsList=_vStrEmpty","")

   <for>("vEmails=0","vEmails < vNumberOfMessages","1")

      <var_oper>(vEmailsList,"%vEmailsList%",STR_APPEND,"vMessages_Subject[vEmails]","", "0")
      <var_oper>(vEmailsList,"%vEmailsList%",STR_APPEND,"%_vKeyNewLine%","", "0")

   <for_end>

   <msg>(-100,-100,"%vEmailsList%","Emails received",1,0,0)

<else>

   <msg>(-100,-100,"There are no e-mails received.","Emails received",1,0,2)

<endif>
```

E-mail POP3: GET E-MAIL - < email_pop3_getmail >() ... [Pro]

E-mail POP3: GET E-MAIL

<email_pop3_getmail>("Connection Id","E-mail number","Variable receiving sender","Variable receiving subject","Variable receiving body")

Available in: Professional edition

This command retrieves defined message waiting on POP3 e-mail account. Before command can be called a connection to the e-mail account needs to be opened using command. In addition, it is always good to check for the e-mail existence using before retrieving the particular e-mail using the command.

#	Parameter name	Parameter description
1	Connection Id	An identifier of connection to the e-mail account. Connection is opened using command.
2	E-mail number	Message number (index) to get. The first e-mail message has number "0", the second one has "1" and so on....
3	Variable receiving sender	Variable that receives "from" field of the e-mail message.
4	Variable receiving subject	Variable that receives "subject" field of the e-mail message.
5	Variable receiving body	Variable that receives "body" field of the e-mail message.

Example (Macro Steps):

```
1   <#> <#> Thic sample shows how to get list of e-mails received on POP3 account.
2   Macro execution: ONLY COMMANDS
3   <#> <#> Connect to POP3 account first:
4   E-mail POP3: CONNECT Server=myemails.com, Login name=myaccount, Password=737qw7523#$,
    Connection Id = POP3Connection, Port = , TLS/SSL =
5   <#> <#> Get number of messages:
6   E-mail POP3: GET LIST Connection Id = "POP3Connection", Variable receiving number of e-mails =
    "vNumOfMessages", Variable array with e-mail senders = "", Variable array with e-mail subjects = ""
7   <#> <#> Are there some messages?
8   IF NUMERIC vNumOfMessages > 0
9       <#> <#> Yes, so let's get the first one:
10      E-mail POP3: GET E-MAIL Connection Id = "POP3Connection", E-mail number = "0", Variable receiving
        sender = "vFrom", Variable receiving subject = "vSubject", Variable receiving body = "vBody"
11      <#> <#> Show the list of messages:
12      Message SHOW "Information" : "Subject: '%vSubject%' From: '%vFrom%' ---------- %vBody%"
        (other parameters: x = -100, y = -100, Window title = Emails received, Buttons = OK, Timeout (seconds) =
        0, Always on top = ).
13  ELSE activate
14      Message SHOW "Error" : "No message is waiting on e-mail account." (other parameters: x = -100, y =
        -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
15  ENDIF
16  <#> <#> Disconnect from the account:
17  E-mail POP3: DISCONNECT Connection Id = POP3Connection
```

Example (Plain Text):

```
<#> Thic sample shows how to get list of e-mails received on POP3 account.
<cmds>

<#> Connect to POP3 account first:
<email_pop3_connect>("myemails.com","myaccount","737qw7523#$","POP3Connection")

<#> Get number of messages:
<email_pop3_getlist>("POP3Connection","vNumOfMessages","","")

<#> Are there some messages?
<if_num>("vNumOfMessages > 0")

  <#> Yes, so let's get the first one:
  <email_pop3_getmail>("POP3Connection","0","vFrom","vSubject","vBody")

    <#> Show the list of messages:
    <msg>(-100,-100,"Subject: '%vSubject%'
```

```
From: '%vFrom%'
-----------------------------
%vBody%","Emails received",1,0,0)

<else>
   <msg>(-100,-100,"No message is waiting on e-mail account.","Message",1,,2)
<endif>

<#> Disconnect from the account:
<email_pop3_disconnect>("POP3Connection")
```

E-mail POP3: DELETE E-MAIL - < email_pop3_deletemail >() ... [Pro]

E-mail POP3: DELETE E-MAIL
<email_pop3_deletemail>("Connection Id","E-mail number")
Available in: Professional edition

This command deletes defined message waiting on POP3 e-mail account. Before command can be called a connection to the e-mail account needs to be opened using command. In addition, it is always good to check for the e-mail existence using before deleting the particular e-mail using the command.

#	Parameter name	Parameter description
1	Connection Id	An identifier of connection to the e-mail account. Connection is opened using command.
2	E-mail number	Message number (index) to delete. The first e-mail message has number "0", the second one has "1" and so on....

Example (Macro Steps):

```
1   <#> <#> Thic sample shows how to delete e-mail from POP3 account.
2   Macro execution: ONLY COMMANDS
3   <#> <#> Connect to POP3 account first:
4   E-mail POP3: CONNECT Server=myemails.com, Login name=myaccount, Password=737qw7523#$,
    Connection Id = POP3Connection, Port = , TLS/SSL =
5   <#> <#> Get number of messages:
6   E-mail POP3: GET LIST Connection Id = "POP3Connection", Variable receiving number of e-mails =
    "vNumOfMessages", Variable array with e-mail senders = "", Variable array with e-mail subjects = ""
7   <#> <#> Are there some messages?
8   IF NUMERIC vNumOfMessages > 0
9       <#> <#> Do we want to delete the message?
10      Message SHOW "Question" : "Do you want to delete e-mail message?" (other parameters: x = -100, y =
        -100, Window title = Emails received, Buttons = Yes and No, Timeout (seconds) = 0, Always on top = ).
11      IF STRING _vMsgButton==YES
12          <#> <#> Yes, so let's delete the first one:
13          E-mail POP3: DELETE E-MAIL Connection Id=POP3Connection, E-mail number=0
14      ENDIF
15  ELSE activate
16      Message SHOW "Error" : "No message is waiting on e-mail account." (other parameters: x = -100, y =
        -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
17  ENDIF
18  <#> <#> Disconnect from the account:
19  E-mail POP3: DISCONNECT Connection Id = POP3Connection
```

Example (Plain Text):

```
<#> Thic sample shows how to delete e-mail from POP3 account.
<cmds>

<#> Connect to POP3 account first:
<email_pop3_connect>("myemails.com","myaccount","737qw7523#$","POP3Connection")

<#> Get number of messages:
<email_pop3_getlist>("POP3Connection","vNumOfMessages","","")

<#> Are there some messages?
<if_num>("vNumOfMessages > 0")

  <#> Do we want to delete the message?
  <msg>(-100,-100,"Do you want to delete e-mail message?","Emails received",2,0,1)
```

```
  <if_str>("_vMsgButton==YES")
    <#> Yes, so let's delete the first one:
    <email_pop3_deletemail>("POP3Connection","0")
  <endif>

<else>
  <msg>(-100,-100,"No message is waiting on e-mail account.","Message",1,,2)
<endif>

<#> Disconnect from the account:
<email_pop3_disconnect>("POP3Connection")
```

E-mail POP3: CONNECT - < email_pop3_connect >() ... [Pro]

E-mail POP3: CONNECT
<email_pop3_connect>("Server","Login name","Password","Connection Id",Port,TLS/SSL)
Available in: Professional edition

This command opens a connection to an e-mail account. This command needs to be called before any other is used.

#	Parameter name	Parameter description
1	Server	Server where e-mail account exist. Example: myemails.com
2	Login name	The e-mail account name.
3	Password	The account password.
4	Connection Id	An identifier of connection to the e-mail account. Other commands use this Id.
5	Port	
6	TLS/SSL	

Example (Macro Steps):

1 <#> <#> Thic sample shows how to get list of e-mails received on POP3 account.

2 **Macro execution: ONLY COMMANDS**

3 <#> <#> Connect to POP3 account first:

4 **E-mail POP3: CONNECT** Server=myemails.com, Login name=myaccount, Password=737qw7523#$, Connection Id = POP3Connection, Port = , TLS/SSL =

5 <#> <#> Do what you need here....

6 <#> <#> Disconnect from the account:

7 **E-mail POP3: DISCONNECT** Connection Id = POP3Connection

Example (Plain Text):

```
<#> Thic sample shows how to get list of e-mails received on POP3 account.
<cmds>

<#> Connect to POP3 account first:
<email_pop3_connect>("myemails.com","myaccount","737qw7523#$","POP3Connection")

<#> Do what you need here....

<#> Disconnect from the account:
<email_pop3_disconnect>("POP3Connection")
```

E-mail POP3: DISCONNECT - < email_pop3_disconnect >() ... [Pro]

E-mail POP3: DISCONNECT
<email_pop3_disconnect>("Connection Id")
Available in: Professional edition

This command closes a connection to an e-mail account. This command needs to be called after all commands are finished.

#	Parameter name	Parameter description
1	Connection Id	An identifier of connection to the e-mail account.

Example (Macro Steps):

1 <#> <#> Thic sample shows how to get list of e-mails received on POP3 account.

2 **Macro execution: ONLY COMMANDS**

3 <#> <#> Connect to POP3 account first:

4 **E-mail POP3: CONNECT** Server=myemails.com, Login name=myaccount, Password=737qw7523#$, Connection Id = POP3Connection, Port = , TLS/SSL =

5 <#> <#> Do what you need here....

6 <#> <#> Disconnect from the account:

7 **E-mail POP3: DISCONNECT** Connection Id = POP3Connection

Example (Plain Text):

```
<#> Thic sample shows how to get list of e-mails received on POP3 account.
<cmds>

<#> Connect to POP3 account first:
<email_pop3_connect>("myemails.com","myaccount","737qw7523#$","POP3Connection")

<#> Do what you need here....

<#> Disconnect from the account:
<email_pop3_disconnect>("POP3Connection")
```

E-mail SMTP SEND MAIL - < email_smtp_sendmail >() ... [Pro]

E-mail SMTP SEND MAIL
<email_smtp_sendmail>("Server","Login name","Password","To","CC","Subject","Message text","Attachment",Port,TLS/SSL,"From (name)","From (e-mail)")
Available in: Professional edition

This command sends an e-mail using SMTP. Unlike , this command does not require any e-mail client to be installed on the computer.

#	Parameter name	Parameter description
1	Server	A SMTP server.
2	Login name	Message number (index) to get. The first e-mail message has number "0", the second one has "1" and so on....
3	Password	Variable that receives "from" field of the e-mail message.
4	To	Variable that receives "subject" field of the e-mail message.
5	CC	Variable that receives "body" field of the e-mail message.
6	Subject	This is a full path to the file to attach to the e-mail message.
7	Message text	Port number to use. If left empty a default port number is used.
8	Attachment	If 0, the SSL is not used otherwise the SSL is used.
9	Port	
10	TLS/SSL	
11	From (name)	
12	From (e-mail)	

Example (Macro Steps):

1 <#> <#> Thic sample shows how send an email.

2 **Macro execution: ONLY COMMANDS**

3 **E-mail SMTP SEND MAIL** Server = "myserver.com", Login name = "myaccount", Password = "mypassword", To = "john@something.com", CC = "", Subject = "Hello John", Message text = "friendly letter from your family. Visit us soon!", Attachment = "", Port = "", TLS/SSL = "No", From (name) = "", From (e-mail) = ""

Example (Plain Text):

<#> Thic sample shows how send an email.
<cmds>

<email_smtp_sendmail>("myserver.com","myaccount","mypassword","john@something.com","","Hello John","friendly letter from your family. Visit us soon!","",,0)

Web FILL FORM - < www_fillform >() ... [Pro]

Web FILL FORM
<www_fillform>("URL","Form fields data","Submit button",Window state,Timeout (seconds),Window handle)
Available in: Professional edition

This command opens a web page with form and fills the form by user pre-defined values. Only Internet Explorer browser is supported.

#	Parameter name	Parameter description
1	URL	Link (URL) to the web page (e.g., "http://www.pitrinec.com/samples/sampleform.htm").
2	Form fields data	The user defined web form data. The user uses "Retrieve URL and Form data" button in this command edit window. It is possible to edit this field manually, though, it will be very hard in most cases. The format is this: {frame index}{form name}{form item type}{form item name}{form item value}{form item selected}.
3	Submit button	A form button that is to be automatically "clicked" after the form is filled. If this field is left empty then the form is not submitted.
4	Window state	The state of the web browser window: 0 - Normal 1 - Maximized 2 - Minimized
5	Timeout (seconds)	Time in seconds the command waits for the web form is loaded and completed.
6	Window handle	

Example (Macro Steps):

1 <#> <#> This simple example shows how to fill web form

2 Macro execution: ONLY COMMANDS

3 Web FILL FORM URL = "http://www.pitrinec.com/samples/sampleform.htm", Timeout (seconds) = "15"

Example (Plain Text):

<#> This simple example shows how to fill web form
<cmds>
<www_fillform>("http://www.pitrinec.com/samples/sampleform.htm","{0}{}{text}{T1}{John}{0}{0}{}{text}{T2}{McKvak}{0}{0}{}{textarea}{S1}{This is my comment:No comment!}{0}{0}{}{}{}{}{0}{0}{}{radio}{R1}{V7}{0}{0}{}{radio}{R1}{V8}{1}{0}{}{radio}{R1}{V9}{0}{0}{}{checkbox}{C1}{ON}{0}{0}{}{checkbox}{C2}{ON}{1}{0}{}{sel-one}{}{1}{0}{0}{}{}{}{}{0}{0}{}{sel-mul}{Dog}{}{0}{0}{}{sel-mul}{Big dog}{}{0}{0}{}{sel-mul}{Big cat}{}{0}{0}{}{sel-mul}{Big mouse}{}{1}{0}{}{sel-mul}{Cat}{}{0}{0}{}{sel-mul}{Mouse}{}{1}{0}{}{submit}{B1}{Submit}{0}","",1,15)

HTML Page Links - < html_page_links >() ... [Pro]

HTML Page Links
<html_page_links>("HTML File",Variable array for links,Variable for number of links,"Login name","Password",Variable array for names)
Available in: Professional edition

This command retrieves all links from the given HTML document. The links are retrieved from all HTML document elements that contain "href" attribute.

#	Parameter name	Parameter description
1	HTML File	(Full) path to the HTML document. It can be either a file on local hardrive or on a shared drive or it can be a web page.
2	Variable array for links	Variable array that receives links.
3	Variable for number of links	Variable that receives number of links retrieved.
4	Login name	Login user name (used to access password protected web page).
5	Password	Login password (used to access password protected web page).
6	Variable array for names	Variable array that receives name of the links. This is the text that is between <a> and .

Example (Macro Steps):

```
1   <#> <#>This example shows how to use HTML Page Links command
2   Macro execution: ONLY COMMANDS
3   HTML Page Links HTML File=https://www.google.com, Variable array for links=vLinks, Variable for number of links=vLinksNum, Login name = , Password = , Variable array for names =
4   Run APPLICATION "notepad.exe" (other parameters: Parameters = , Folder path = , Window state = Normal). Macro execution waits for application to finish up to "0" seconds.
5   WAIT FOR Object = "WIN", Event = "OPEN", Parameter = "[* - Notepad|Notepad|#0|#119]", Timeout (seconds) = "15", Exact = "0"
6   Window ACTIVATE bring "[* - Notepad|Notepad|#0|#119]" window to top (other parameters: Match = Partial, Window state = Normal, %p4_name = )
7   IF WINDOW "[* - Notepad|Notepad|#0|#119]" Is Active (Match=Partial)
8       Repeat steps UNTIL "%i%<vLinksNum" (Counter variable initial value = "i=0", Counter loop increment = "1")
9           Variable INSERT to active application "%vLinks[i]%%_vKeyReturn%", Text insertion method="Default (as defined in settings)"
10      Repeat steps END
11  ENDIF
```

Example (Plain Text):

```
<#>This example shows how to use HTML Page Links command
<cmds>
<html_page_links>("https://www.google.com",vLinks,vLinksNum,"","")
<execappex>("notepad.exe","","",0,0)
<waitfor>("WIN","OPEN","[* - Notepad|Notepad|#0|#119]",15,0)
<actwin>("[* - Notepad|Notepad|#0|#119]",0,0)
<if_win>("[* - Notepad|Notepad|#0|#119]","ACT",0)
   <for>("i=0","%i%<vLinksNum","1")
      <varout>("%vLinks[i]%%_vKeyReturn%",0)
   <for_end>
<endif>
```

ODBC

OPEN - < odbc_open >() ... [Pro]

ODBC OPEN
<odbc_open>("Connection string",Variable for result)
Available in: Professional edition

This command opens a database using an ODBC driver. Note: The ODBC driver must be installed on the computer otherwise the command will fail. The Macro Toolworks (Perfect Keyboard) is a 32-bit application which requires a 32-bit ODBC driver.

#	Parameter name	Parameter description
1	Connection string	ODBC connection string. The connection string format depends on the type of the database. Get the format of the ODBC connection string from the database software provider or get the connection string formats from other sources, for example, https://www.connectionstrings.com Example for Microsoft Access: "Driver={MICROSOFT ACCESS DRIVER (*.mdb, *.accdb)}; Dbq=c:\users\John Jimm\\documents\\MyDatabase.accdb;"
2	Variable for result	Variable that receives the handle of the opened database. This handle is then used in other ODBC commands as a parameter identifying the database.

Example (Macro Steps):

```
1   <#> <#> This example shows how to open and close a Microsoft Access database
2   Macro execution: ONLY COMMANDS
3   <#> <#> Show a form that lets a user to select the database file
4   Form FIELD "Microsoft Access database:" of type "File path" (Default value=*.accdb, Variable to save field
    value=vDbFile, Form identifier=f1)
5   Form OPEN "f1", Window title="Select Microsoft Access database to open"
6   IF %_vCanceled%==1
7       <#> <#> User canceled the selection form, no database file is selected
8       Macro EXIT
9   ENDIF
10  IF FILE "%vDbFile%" Not Exist ()
11      <#> <#> User entered a file that does not exist
12      Message SHOW "Error" : "The file '%vDbFile%' was not found." (other parameters: x = -100, y = -100,
        Window title = , Buttons = OK, Timeout (seconds) = 0, Always on top = No).
13      Macro EXIT
14  ENDIF
15  <#> <#> Build the Microsoft Access connection string
16  Variable SET "vConnectionString=Driver={MICROSOFT ACCESS DRIVER (*.mdb, *.accdb)};
    Dbq=%vDbFile%;", Message text=""
17  <#> <#> Open the database
18  ODBC OPEN Connection string=%vConnectionString%, Variable for result=vDbOpen
19  IF %vDbOpen%!=0
20      Message SHOW "Information" : "Database was sucessfully open." (other parameters: x = -100, y = -100,
        Window title = , Buttons = OK, Timeout (seconds) = 0, Always on top = No).
21      <#> <#> Close the database
22      ODBC CLOSE Database handle = %vDbOpen%
23  ENDIF
```

Example (Plain Text):

```
<#> This example shows how to open and close a Microsoft Access database
<cmds>

<#> Show a form that lets a user to select the database file
<form_item>("f1","Microsoft Access database:","EDIT_FILE","*.accdb","vDbFile",1)
<form_show>("f1","Select Microsoft Access database to open","",0,500,0,,,1,1)
```

```
<if>("%_vCanceled%==1")
   <#> User canceled the selection form, no database file is selected
   <exitmacro>
<endif>

<if_file>("%vDbFile%","NOTEXIST","")
   <#> User entered a file that does not exist
   <msg>(-100,-100,"The file '%vDbFile%' was not found.","",1,0,2,0)
   <exitmacro>
<endif>

<#> Build the Microsoft Access connection string
<varset>("vConnectionString=Driver={MICROSOFT ACCESS DRIVER (*.mdb, *.accdb)}; Dbq=%vDbFile%;","")

<#> Open the database
<odbc_open>("%vConnectionString%",vDbOpen)
<if>("%vDbOpen%!=0")
   <msg>(-100,-100,"Database was sucessfully open.","",1,0,0,0)

   <#> Close the database
   <odbc_close>(%vDbOpen%)
<endif>
```

CLOSE - < odbc_close >() ... [Pro]

ODBC CLOSE
<odbc_close>(Database handle)
Available in: Professional edition

This command closes a database previosly open using the "ODBC OPEN" command.

#	Parameter name	Parameter description
1	Database handle	Database handle. This is the value that was returned from ODBC OPEN command.

Example (Macro Steps):

```
1   <#> <#> This example shows how to open and close a Microsoft Access database
2   Macro execution: ONLY COMMANDS
3   <#> <#> Show a form that lets a user to select the database file
4   Form FIELD "Microsoft Access database:" of type "File path" (Default value=*.accdb, Variable to save field value=vDbFile, Form identifier=f1)
5   Form OPEN "f1", Window title="Select Microsoft Access database to open"
6   IF %_vCanceled%==1
7       <#> <#> User canceled the selection form, no database file is selected
8       Macro EXIT
9   ENDIF
10  IF FILE "%vDbFile%" Not Exist ()
11      <#> <#> User entered a file that does not exist
12      Message SHOW "Error" : "The file '%vDbFile%' was not found." (other parameters: x = -100, y = -100, Window title = , Buttons = OK, Timeout (seconds) = 0, Always on top = No).
13      Macro EXIT
14  ENDIF
15  <#> <#> Build the Microsoft Access connection string
16  Variable SET "vConnectionString=Driver={MICROSOFT ACCESS DRIVER (*.mdb, *.accdb)}; Dbq=%vDbFile%;", Message text=""
17  <#> <#> Open the database
18  ODBC OPEN Connection string=%vConnectionString%, Variable for result=vDbOpen
19  IF %vDbOpen%!=0
20      Message SHOW "Information" : "Database was sucessfully open." (other parameters: x = -100, y = -100, Window title = , Buttons = OK, Timeout (seconds) = 0, Always on top = No).
21      <#> <#> Close the database
22      ODBC CLOSE Database handle = %vDbOpen%
23  ENDIF
```

Example (Plain Text):

```
<#> This example shows how to open and close a Microsoft Access database
<cmds>

<#> Show a form that lets a user to select the database file
<form_item>("f1","Microsoft Access database:","EDIT_FILE","*.accdb","vDbFile",1)
<form_show>("f1","Select Microsoft Access database to open","",0,500,0,,,1,1)
```

```
<if>("%_vCanceled%==1")
   <#> User canceled the selection form, no database file is selected
   <exitmacro>
<endif>

<if_file>("%vDbFile%","NOTEXIST","")
   <#> User entered a file that does not exist
   <msg>(-100,-100,"The file '%vDbFile%' was not found.","",1,0,2,0)
   <exitmacro>
<endif>

<#> Build the Microsoft Access connection string
<varset>("vConnectionString=Driver={MICROSOFT ACCESS DRIVER (*.mdb, *.accdb)}; Dbq=%vDbFile%;","")

<#> Open the database
<odbc_open>("%vConnectionString%",vDbOpen)
<if>("%vDbOpen%!=0")
   <msg>(-100,-100,"Database was sucessfully open.","",1,0,0,0)

   <#> Close the database
   <odbc_close>(%vDbOpen%)
<endif>
```

Execute SQL - < odbc_exec_sql >() ... [Pro]

ODBC Execute SQL
<odbc_exec_sql>(Database handle,"Execute SQL")
Available in: Professional edition

This command executes an SQL command on a database previosly open using the "ODBC OPEN" command.

#	Parameter name	Parameter description
1	Database handle	Database handle. This is the value that was returned from ODBC OPEN command.
2	Execute SQL	SQL command. For example: UPDATE BookTable SET Count=5 WHERE ID=2

Example (Macro Steps):

```
1   <#> <#> This example shows how to execute an SQL command on opened Microsoft Access database
2   Macro execution: ONLY COMMANDS
3   <#> <#> Show a form that lets a user to select the database file
4   Form FIELD "Microsoft Access database:" of type "File path" (Default value=*.accdb, Variable to save field value=vDbFile, Form identifier=f1)
5   Form OPEN "f1", Window title="Select Microsoft Access database to open"
6   IF %_vCanceled%==1
7       <#> <#> User canceled the selection form, no database file is selected
8       Macro EXIT
9   ENDIF
10  IF FILE "%vDbFile%" Not Exist ()
11      <#> <#> User entered a file that does not exist
12      Message SHOW "Error" : "The file '%vDbFile%' was not found." (other parameters: x = -100, y = -100, Window title = , Buttons = OK, Timeout (seconds) = 0, Always on top = No).
13      Macro EXIT
14  ENDIF
15  <#> <#> Build the Microsoft Access connection string
16  Variable SET "vConnectionString=Driver={MICROSOFT ACCESS DRIVER (*.mdb, *.accdb)}; Dbq=%vDbFile%;", Message text=""
17  <#> <#> Open the database
18  ODBC OPEN Connection string=%vConnectionString%, Variable for result=vDbOpen
19  IF %vDbOpen%!=0
20      <#> <#> Set John's age to 33
21      ODBC Execute SQL Database handle=%vDbOpen%, Execute SQL=UPDATE TestTable1 SET AGE=33 WHERE FRIEND='John'
22      <#> <#> Close the database
23      ODBC CLOSE Database handle = %vDbOpen%
24  ENDIF
```

Example (Plain Text):

```
<#> This example shows how to execute an SQL command on opened Microsoft Access database
<cmds>
```

```
<#> Show a form that lets a user to select the database file
<form_item>("f1","Microsoft Access database:","EDIT_FILE","*.accdb","vDbFile",1)
<form_show>("f1","Select Microsoft Access database to open","",0,500,0,,,1,1)

<if>("%_vCanceled%==1")
   <#> User canceled the selection form, no database file is selected
   <exitmacro>
<endif>

<if_file>("%vDbFile%","NOTEXIST","")
   <#> User entered a file that does not exist
   <msg>(-100,-100,"The file '%vDbFile%' was not found.","",1,0,2,0)
   <exitmacro>
<endif>

<#> Build the Microsoft Access connection string
<varset>("vConnectionString=Driver={MICROSOFT ACCESS DRIVER (*.mdb, *.accdb)}; Dbq=%vDbFile%;","")

<#> Open the database
<odbc_open>("%vConnectionString%",vDbOpen)
<if>("%vDbOpen%!=0")

   <#> Set John's age to 33
   <odbc_exec_sql>(%vDbOpen%,"UPDATE TestTable1 SET AGE=33 WHERE FRIEND='John'")

   <#> Close the database
   <odbc_close>(%vDbOpen%)
<endif>
```

Select SQL - < odbc_select >() ... [Pro]

ODBC Select SQL

<odbc_select>(Database handle,"Select SQL",Variable for result)
Available in: Professional edition

This command executes an SQL select command on a database previosly open using the "ODBC OPEN" command. The selected data set then can be retrived using ODBC Select GET command and enumerated using ODBC Select NEXT command.

#	Parameter name	Parameter description
1	Database handle	Database handle. This is the value that was returned from ODBC OPEN command.
2	Select SQL	Select SQL command. For example: UPDATE BookTable SET Count=5 WHERE ID=2
3	Variable for result	Variable that receives identifier of the selected data set. This value is used in the ODBC Select GET and ODBC Select NEXT commands. The value is non-zero if the data set is not empty.

Example (Macro Steps):

```
1  <#> <#> This example shows how to get values from the selected data set of the Microsoft Access database
2  Macro execution: ONLY COMMANDS
3  <#> <#> Show a form that lets a user to select the database file
4  Form FIELD "Microsoft Access database:" of type "File path" (Default value=*.accdb, Variable to save field value=vDbFile, Form identifier=f1)
5  Form OPEN "f1", Window title="Select Microsoft Access database to open"
6  IF %_vCanceled%==1
7      <#> <#> User canceled the selection form, no database file is selected
8      Macro EXIT
9  ENDIF
10 IF FILE "%vDbFile%" Not Exist ()
11     <#> <#> User entered a file that does not exist
12     Message SHOW "Error" : "The file '%vDbFile%' was not found." (other parameters: x = -100, y = -100, Window title = , Buttons = OK, Timeout (seconds) = 0, Always on top = No).
13     Macro EXIT
14 ENDIF
15 <#> <#> Build the Microsoft Access connection string
16 Variable SET "vConnectionString=Driver={MICROSOFT ACCESS DRIVER (*.mdb, *.accdb)}; Dbq=%vDbFile%;", Message text=""
17 <#> <#> Open the database
18 ODBC OPEN Connection string=%vConnectionString%, Variable for result=vDbOpen
19 IF %vDbOpen%!=0
20     <#> <#> Select whole table
21     ODBC Select SQL (Database handle = "%vDbOpen%", Select SQL = "SELECT * FROM TestTable1 ", Variable for result = "vDataSet")
22     <#> <#> Cycle in the data set
23     Repeat steps UNTIL "%vDataSet%!=0" (Counter variable initial value = "", Counter loop increment = "")
24         <#> <#> Get data from the record
25         ODBC Select GET Database handle = "%vDbOpen%", Select handle = "%vDataSet%", Field name = "FRIEND", Variable for result = "vFriend"
26         ODBC Select GET Database handle = "%vDbOpen%", Select handle = "%vDataSet%", Field name = "AGE", Variable for result = "vAge"
```

Example (Plain Text):

```
<#> This example shows how to get values from the selected data set of the Microsoft Access database
<cmds>

<#> Show a form that lets a user to select the database file
<form_item>("f1","Microsoft Access database:","EDIT_FILE","*.accdb","vDbFile",1)
<form_show>("f1","Select Microsoft Access database to open","",0,500,0,,,1,1)

<if>("%_vCanceled%==1")
  <#> User canceled the selection form, no database file is selected
  <exitmacro>
<endif>

<if_file>("%vDbFile%","NOTEXIST","")
  <#> User entered a file that does not exist
  <msg>(-100,-100,"The file '%vDbFile%' was not found.","",1,0,2,0)
  <exitmacro>
<endif>

<#> Build the Microsoft Access connection string
<varset>("vConnectionString=Driver={MICROSOFT ACCESS DRIVER (*.mdb, *.accdb)}; Dbq=%vDbFile%;","")

<#> Open the database
<odbc_open>("%vConnectionString%",vDbOpen)
<if>("%vDbOpen%!=0")

  <#> Select whole table
  <odbc_select>(%vDbOpen%,"SELECT * FROM TestTable1 ",vDataSet)<#>

  <#> Cycle in the data set
  <for>("","%vDataSet%!=0","")

    <#> Get data from the record
    <odbc_select_get>(%vDbOpen%,%vDataSet%,"FRIEND",vFriend)
    <odbc_select_get>(%vDbOpen%,%vDataSet%,"AGE",vAge)
    <msg>(-100,-100,"Friend=%vFriend%
Age=%vAge%","",1,0,0,0)

    <#> Move to next record
    <odbc_select_next>(%vDbOpen%,vDataSet)

  <for_end>

  <#> Close the database
  <odbc_close>(%vDbOpen%)
<endif>
```

Select GET - < odbc_select_get >() ... [Pro]

ODBC Select GET
<odbc_select_get>(Database handle,Select handle,"Field name",Variable for result)
Available in: Professional edition

This command retrieves data from a data set previously selected by the ODBC Select SQL command.

#	Parameter name	Parameter description
1	Database handle	Database handle. This is the value that was returned from ODBC OPEN command.
2	Select handle	Data set handle. This is the value that was returned from the ODBC Select SQL command.
3	Field name	Name of the field to be retrieved.
4	Variable for result	Name of the variable that receives the field value.

Example (Macro Steps):

```
<#> <#> This example shows how to get values from the selected data set of the Microsoft Access database

Macro execution: ONLY COMMANDS

<#> <#> Show a form that lets a user to select the database file

Form FIELD "Microsoft Access database:" of type "File path" (Default value=*.accdb, Variable to save field value=vDbFile, Form identifier=f1)

Form OPEN "f1", Window title="Select Microsoft Access database to open"

IF %_vCanceled%==1

    <#> <#> User canceled the selection form, no database file is selected

    Macro EXIT

ENDIF

IF FILE "%vDbFile%" Not Exist ()

    <#> <#> User entered a file that does not exist

    Message SHOW "Error" : "The file '%vDbFile%' was not found." (other parameters: x = -100, y = -100, Window title = , Buttons = OK, Timeout (seconds) = 0, Always on top = No).

    Macro EXIT

ENDIF

<#> <#> Build the Microsoft Access connection string

Variable SET "vConnectionString=Driver={MICROSOFT ACCESS DRIVER (*.mdb, *.accdb)}; Dbq=%vDbFile%;", Message text=""

<#> <#> Open the database

ODBC OPEN Connection string=%vConnectionString%, Variable for result=vDbOpen

IF %vDbOpen%!=0

    <#> <#> Select whole table

    ODBC Select SQL (Database handle = "%vDbOpen%", Select SQL = "SELECT * FROM TestTable1 ", Variable for result = "vDataSet")

    <#> <#> Cycle in the data set

    Repeat steps UNTIL "%vDataSet%!=0" (Counter variable initial value = "", Counter loop increment = "")

        <#> <#> Get data from the record

        ODBC Select GET Database handle = "%vDbOpen%", Select handle = "%vDataSet%", Field name = "FRIEND", Variable for result = "vFriend"

        ODBC Select GET Database handle = "%vDbOpen%", Select handle = "%vDataSet%", Field name = "AGE", Variable for result = "vAge"
```

Example (Plain Text):

```
<#> This example shows how to get values from the selected data set of the Microsoft Access database
<cmds>

<#> Show a form that lets a user to select the database file
<form_item>("f1","Microsoft Access database:","EDIT_FILE","*.accdb","vDbFile",1)
<form_show>("f1","Select Microsoft Access database to open","",0,500,0,,,1,1)

<if>("%_vCanceled%==1")
  <#> User canceled the selection form, no database file is selected
  <exitmacro>
<endif>

<if_file>("%vDbFile%","NOTEXIST","")
  <#> User entered a file that does not exist
  <msg>(-100,-100,"The file '%vDbFile%' was not found.","",1,0,2,0)
  <exitmacro>
<endif>

<#> Build the Microsoft Access connection string
<varset>("vConnectionString=Driver={MICROSOFT ACCESS DRIVER (*.mdb, *.accdb)}; Dbq=%vDbFile%;","")

<#> Open the database
<odbc_open>("%vConnectionString%",vDbOpen)
<if>("%vDbOpen%!=0")

  <#> Select whole table
  <odbc_select>(%vDbOpen%,"SELECT * FROM TestTable1 ",vDataSet)<#>

  <#> Cycle in the data set
  <for>("","%vDataSet%!=0","")

    <#> Get data from the record
    <odbc_select_get>(%vDbOpen%,%vDataSet%,"FRIEND",vFriend)
    <odbc_select_get>(%vDbOpen%,%vDataSet%,"AGE",vAge)
    <msg>(-100,-100,"Friend=%vFriend%
Age=%vAge%","",1,0,0,0)

    <#> Move to next record
    <odbc_select_next>(%vDbOpen%,vDataSet)

  <for_end>

  <#> Close the database
  <odbc_close>(%vDbOpen%)
<endif>
```

Select NEXT - < odbc_select_next >() ... [Pro]

ODBC Select NEXT
<odbc_select_next>(Database handle,Select handle)
Available in: Professional edition

This command moves the currently selected record in the data set to the next record.

#	Parameter name	Parameter description
1	Database handle	Database handle. This is the value that was returned from ODBC OPEN command.
2	Select handle	Name of the variable that contains the data set handle. This is typically the save variable that is use as a last parameter in the ODBC Select SQL command. When the end of the data set is reached then the value of the variable is set to 0.

Example (Macro Steps):

```
<#> <#> This example shows how to get values from the selected data set of the Microsoft Access database

Macro execution: ONLY COMMANDS

<#> <#> Show a form that lets a user to select the database file

Form FIELD "Microsoft Access database:" of type "File path" (Default value=*.accdb, Variable to save field value=vDbFile, Form identifier=f1)

Form OPEN "f1", Window title="Select Microsoft Access database to open"

IF %_vCanceled%==1

    <#> <#> User canceled the selection form, no database file is selected

    Macro EXIT

ENDIF

IF FILE "%vDbFile%" Not Exist ()

    <#> <#> User entered a file that does not exist

    Message SHOW "Error" : "The file '%vDbFile%' was not found." (other parameters: x = -100, y = -100, Window title = , Buttons = OK, Timeout (seconds) = 0, Always on top = No).

    Macro EXIT

ENDIF

<#> <#> Build the Microsoft Access connection string

Variable SET "vConnectionString=Driver={MICROSOFT ACCESS DRIVER (*.mdb, *.accdb)}; Dbq=%vDbFile%;", Message text=""

<#> <#> Open the database

ODBC OPEN Connection string=%vConnectionString%, Variable for result=vDbOpen

IF %vDbOpen%!=0

    <#> <#> Select whole table

    ODBC Select SQL (Database handle = "%vDbOpen%", Select SQL = "SELECT * FROM TestTable1 ", Variable for result = "vDataSet")

    <#> <#> Cycle in the data set

    Repeat steps UNTIL "%vDataSet%!=0" (Counter variable initial value = "", Counter loop increment = "")

        <#> <#> Get data from the record

        ODBC Select GET Database handle = "%vDbOpen%", Select handle = "%vDataSet%", Field name = "FRIEND", Variable for result = "vFriend"

        ODBC Select GET Database handle = "%vDbOpen%", Select handle = "%vDataSet%", Field name = "AGE", Variable for result = "vAge"
```

Example (Plain Text):

```
<#> This example shows how to get values from the selected data set of the Microsoft Access database
<cmds>

<#> Show a form that lets a user to select the database file
<form_item>("f1","Microsoft Access database:","EDIT_FILE","*.accdb","vDbFile",1)
<form_show>("f1","Select Microsoft Access database to open","",0,500,0,,,1,1)

<if>("%_vCanceled%==1")
  <#> User canceled the selection form, no database file is selected
  <exitmacro>
<endif>

<if_file>("%vDbFile%","NOTEXIST","")
  <#> User entered a file that does not exist
  <msg>(-100,-100,"The file '%vDbFile%' was not found.","",1,0,2,0)
  <exitmacro>
<endif>

<#> Build the Microsoft Access connection string
<varset>("vConnectionString=Driver={MICROSOFT ACCESS DRIVER (*.mdb, *.accdb)}; Dbq=%vDbFile%;","")

<#> Open the database
<odbc_open>("%vConnectionString%",vDbOpen)
<if>("%vDbOpen%!=0")

  <#> Select whole table
  <odbc_select>(%vDbOpen%,"SELECT * FROM TestTable1 ",vDataSet)<#>

  <#> Cycle in the data set
  <for>("","%vDataSet%!=0","")

    <#> Get data from the record
    <odbc_select_get>(%vDbOpen%,%vDataSet%,"FRIEND",vFriend)
    <odbc_select_get>(%vDbOpen%,%vDataSet%,"AGE",vAge)
    <msg>(-100,-100,"Friend=%vFriend%
Age=%vAge%","",1,0,0,0)

    <#> Move to next record
    <odbc_select_next>(%vDbOpen%,vDataSet)

  <for_end>

  <#> Close the database
  <odbc_close>(%vDbOpen%)
<endif>
```

Run/Execute

MACRO - < run >() ... [Pro]

Run MACRO
<run>("Macro","Parameters","Time to wait")
Available in: Professional edition

Runs other macro (specified by name). It is possible to pass the called macro a parameter and it is also possible to receive a returns some data then the data are available through "_vMacroResult" system variable right after the command finishes exec

Note below parameters specific to "Clipboard macros".

#	Parameter name	Parameter description
1	Macro	Name or ID (showing in the "general" tab in macro definition view) of the macro to run.
2	Parameters	Parameter that will be passed to the remote macro. The called macro can retrieve the paramet - If the macro called is "Clipboard macro" with a text (including rich text) content then it is pos other text specified in the parameter. The syntax is: replace_what:EXISTING_TEXT;replace_by:NEW_TEXT; Example: replace_what:rep01;replace_by:%vCustomerName%;replace_what:rep02;replace_by:%vHelpP Where EXISTING_TEXT is a text in the "Clipboard macro" and NEW_TEXT is a new text (can l before the "Clipboard macro" is pasted. There can be multiple replace_what;replace_by; pairs in the parameter. It is recommended to like "%rep_01%") as a text to replace since different rich text formats in "Clipboard macro" en differently and replacement could fail. This feature can be used to create nicely formatted templates of documents and e-mails that c - If the macro called is "Clipboard macro" then it is possible to specify here "load_only" param copied to clipboard but not pasted to an application.
3	Time to wait	Time in milliseconds to wait before next command is executed. If this parameter is empty ther

Example (Macro Steps):

```
1   <#> <#> This example shows how to call a macro called "ToUpper" using co
2   <#> <#> We pass parameter "lowercase string" that the "ToUpper"
3   <#> <#> macro converts to uper case and passes back as a result.
4   Macro execution: ONLY COMMANDS
5   Run MACRO ToUpper, Parameters = lowercase string,Time to wait =
6   Message SHOW "Information" : "%_vMacroResult%" (other parameters: x = -100, y
    Timeout (seconds) = 0, Always on top = ).
```

Example (Plain Text):

```
<#> This example shows how to call a macro called "ToUpper" using <run> command.
<#> We pass parameter "lowercase string" that the "ToUpper"
<#> macro converts to uper case and passes back as a result.
```

```
<#>
<cmds>
<run>("ToUpper","lowercase string")
<msg>(-100,-100,"%_vMacroResult%","And the result is",1,0,0)
```

SELECTED MACRO - < listbox >() ... [Pro]

Run SELECTED MACRO
<listbox>("Window Title","Macro 1 name", "Macro 1 description", "Macro 2 name", "Macro 2 description", ...)
Available in: Professional edition

This command displays a list of macros a user can choose from. The selected macro is executed and the control is returned to the calling macro again. If user clicks "Cancel" button the _vCanceled system variable is set to 1 (otherwise it is 0) to allow to distinguish between situations when OK or Cancel button was clicked.

#	Parameter name	Parameter description
1	Window title	Title of the macro selection window.
2	Macro	Name of the macro (or macro Id) to run if selected.
3	Description	Description of the macro so that user knows what to select.

Example (Macro Steps):

1 <#> <#> This macro demonstrates how to create a list of macros

2 <#> <#> a user can select from to run a particular macro

3 **Macro execution: ONLY COMMANDS**

4 **Run SELECTED MACRO** "Window title = Select your option", macros to select from: _macro1 - 1. Macro; _macro2 - 2. Macro--d.e.f.a.u.l.t--; _macro3 - 3. Macro

Example (Plain Text):

```
<#> This macro demonstrates how to create a list of macros
<#> a user can select from to run a particular macro
<#>
<cmds>
<listbox>("Select your option","_macro1","1. Macro","_macro2","2. Macro--d.e.f.a.u.l.t--","_macro3","3. Macro")
```

.MCR FILE - < extmacro >() ... [Pro]

Run .MCR FILE
<extmacro>("File path",Text insertion method,"Parameters","Time to wait")
Available in: Professional edition

Runs macro saved in an external text file. It is possible to pass the external macro a parameter and it is also possible to receive a return data from the macro. If the macro returns some data then the data are available through "_vMacroResult" system variable right after the command finishes execution.

#	Parameter name	Parameter description
1	File path	Name of the macro to run. (The macro name should be unique if it is run from other macros using command).
2	Text insertion method	If 1, the text from the MacroFile is inserted to the active window through the clipboard. (This option is intended only for plain text with no commands.) If 0, the macro from the MacroFile is played back the same way as it was regular macro.
3	Parameters	Parameter that will be passed to the remote macro. The called macro can retrieve the parameter data from "_vMacroParameter" system variable.
4	Time to wait	Time in milliseconds to wait before next command is executed. If this parameter is empty then a default wait time is used.

Example (Macro Steps):

1 <#> <#> This macro starts macro from external file.

2 Macro execution: ONLY COMMANDS

3 **Run .MCR FILE** run macro "%_vMacroFileFolder%Samples\Simple form example.mcr" with parameter "". Text insertion method is "Keystrokes sequence". Parameters =

Example (Plain Text):

```
<#> This macro starts macro from external file.
<#>
<cmds>
<extmacro>("%_vMacroFileFolder%Samples\Simple form example.mcr",0)
```

APPLICATION - < execappex >() ... [Free]

Run APPLICATION
<execappex>("File path","Parameters","Folder path",Window state,Time to wait)
Available in: Free edition

Starts an application or opens a file or web page.

#	Parameter name	Parameter description
1	File path	(Full) path to the application's file (e.g., "c:\program files\app\app.exe").
2	Parameters	List of parameters the application should be started with (e.g., "/n /g").
3	Folder path	Start up directory. The directory is made current before the application is started.
4	Window state	The state of the window: 0 - Normal 1 - Maximized 2 - Minimized 3 - Hidden
5	Time to wait	Must be 0.

Example (Macro Steps):

1 <#> <#> This command starts Notepad maximized.

2 **Macro execution: ONLY COMMANDS**

3 **Run APPLICATION** "notepad.exe" (other parameters: Parameters = , Folder path = , Window state = Maximize). Macro execution waits for application to finish up to "0" seconds.

Example (Plain Text):

```
<#> This command starts Notepad maximized.
<#>
<cmds>
<execappex>("notepad.exe","","",1,0)
```

EXTERNAL SCRIPT FILE - < script_file >() ... [Pro]

Run EXTERNAL SCRIPT FILE
<script_file>("File",Wait until finished)
Available in: Professional edition

Executes a script from external file.

#	Parameter name	Parameter description
1	File	Script file to run. There are these file types supported: .bss - Basic Script file .bsx - encoded Basic Script file .vbs - VB Script file .js - Java Script file The file should be either full path (e.g., "c:\myscripts\script1.bss") or just file name (e.g., "script.bss"). Partial or relative paths (e.g., "..\scripts\script.bss") are not supported at this time. All the scripts that you call using just the name (e.g., "script.bss") must be saved in the "scripts" directory. In the "Basic Script" scripts, it is possible to use '#Uses "some-file" directives that allows to use symbols defined in other modules. In the "uses" directives, both BSS and BSX files can be used (they also have to be in "scripts" directory or full path must be supplied).
2	Wait until finished	If equal to 1, the macro waits until script execution is finished. If equal to 0, the macro continues execution without waiting for script to finish.

Example (Macro Steps):

```
1    <#> <#> This macro runs selected script
2    Macro execution: ONLY COMMANDS
3    Variable OPERATION "SELECT_FILE" (Variable for result = vScript, Input text/variable = *.bss, Parameter 1 = Select Script To Run, Parameter 2 = , Parameter 3 = 0)
4    IF STRING _vCanceled==1
5        Macro EXIT
6    ENDIF
7    Run EXTERNAL SCRIPT FILE "vScript", Wait until finished="No"
```

Example (Plain Text):

```
<#> This macro runs selected script
<#>
<cmds>
<var_oper>(vScript,"*.bss",SELECT_FILE,"Select Script To Run","", "0")
<if_str>("_vCanceled==1") <exitmacro> <endif>

<script_file>("vScript",0)
```

FILE CONTEXT MENU COMMAND - < run_ctxcommand >() ... [Pro]

Run FILE CONTEXT MENU COMMAND
<run_ctxcommand>("File path",Menu command,Unused,Unused)
Available in: Professional edition

This command performs specified Windows shell "context menu command".
Note: When a user right-clicks on a file in Windows Explorer a menu so called "context menu" is shown. There are listed several commands a user can perform on the file (like "Open", "Delete", "Copy", etc.).

#	Parameter name	Parameter description
1	File path	(Full) path to the file (e.g., "c:\mydocuments\doc1.doc") to run a command on. Can be a static text or variable containing text.
2	Menu command	Menu command as shown on the menu. For example: Open, Copy, Delete, etc. If the command is in submenu then use "\" to separate submenu. Example: Send To\My Documents, WinZip\Add to Zip file..., etc. Commands can contain also wildcards (* and ?). Example: WinZ*\Add
3	Unused	Must be 0.
4	Unused	Must be 0.

Example (Macro Steps):

1 <#> <#> This command opens property dialog of "C:\temp" directory

2 **Macro execution: ONLY COMMANDS**

3 **Run FILE CONTEXT MENU COMMAND** "Properties" on file "c:\temp"

Example (Plain Text):

```
<#> This command opens property dialog of "C:\temp" directory
<#>
<cmds>
<run_ctxcommand>("c:\temp",Properties,0,0)
```

EXTERNAL COMMAND - < extcmd >() ... [Pro]

Run EXTERNAL COMMAND
<extcmd>("Command executable","Parameters",Timeout (seconds),Variable receiving result)
Available in: Professional edition

This command executes an external command. The external commands allows to extend the macro language capabilities by adding new commands. The external command can be any executable file that takes parameters (optionally) and prints result to standard output. The macro engine takes the printed result from standard output and assigns it to user-defined macro language result variable.

Note: The external command can be written in VBS, for example. However, the VBS engine outputs some copyright information at the start of the script execution. If the user does not want to have such the information as a part of own script result then the user can put this line at the begin of VBS script:

Wscript.StdOut.Write ">>>cmd_results>>>"

This line tells the macro engine where actual results start and the macro engine then ignores the output printed before.

#	Parameter name	Parameter description
1	Command executable	(Full) path to an executable file or just a name with ".ec...." extension (for example, "SimplePing.ec.vbs"). If the full path is not supplied then the executable file must be located in "ExternalCommands" sub-folder in installation folder or in the same folder where calling macro file is located.
2	Parameters	Parameters that will be passed to the external command. The number of the parameters depends on the external command.
3	Timeout (seconds)	Timeout in seconds. If the external command does not finishes its execution in the defined timeout then the command fails.
4	Variable receiving result	This is the user-defined variable that will receive the external command result.

Example (Macro Steps):

```
1   <#> <#> This command pings to the www.pitrinec.com web site
2   Macro execution: ONLY COMMANDS
3   Run EXTERNAL COMMAND Command executable = "SimplePing.ec.vbs", Parameters = "www.pitrinec.com",
    Timeout (seconds) = "15", Variable receiving result = "vResult"
4   IF STRING vResult==1
5       Message SHOW "Information" : "Ping OK." (other parameters: x = -100, y = -100, Window title = Message,
        Buttons = OK, Timeout (seconds) = 0, Always on top = ).
6   ELSE activate
7       Message SHOW "Information" : "Ping failed." (other parameters: x = -100, y = -100, Window title =
        Message, Buttons = OK, Timeout (seconds) = 0, Always on top = ).
8   ENDIF
```

Example (Plain Text):

```
<#> This command pings to the www.pitrinec.com web site
<cmds>
<extcmd>("SimplePing.ec.vbs","www.pitrinec.com",15,vResult)
<if_str>("vResult==1")
   <msg>(-100,-100,"Ping OK.","Message",1,0,0)
<else>
   <msg>(-100,-100,"Ping failed.","Message",1,0,0)
<endif>
```

System

Screensaver START - < scrsavestart > ... [Pro]

Screensaver START
<scrsavestart>
Available in: Professional edition

This command starts screensaver.

Example (Macro Steps):

1	<#> <#> This macro starts screensaver
2	**Screensaver START**

Example (Plain Text):

```
<#> This macro starts screensaver
<scrsavestart>
```

Set system TIME - < setsystime >() ... [Pro]

Set system TIME
<setsystime>(Hour,Minute,Second)
Available in: Professional edition

Sets system time. Note: This command requires administrator privileges.

#	Parameter name	Parameter description
1	Hour	Hour of the newly set system time.
2	Minute	Minute of the newly set system time.
3	Second	Second of the newly set system time.

Example (Macro Steps):

1 <#> <#> This macro will set system time to 05:00:00

2 **Macro execution: ONLY COMMANDS**

3 Set system TIME Hour=5, Minute=0, Second=0

Example (Plain Text):

```
<#> This macro will set system time to 05:00:00
<#>
<cmds>
<setsystime>(5,0,0)
```

Set system DATE - < setsysdate >() ... [Pro]

Set system DATE
<setsysdate>(Year,Month,Day)
Available in: Professional edition

Sets system date. Note: Thos cpmmand requires administrator privileges.

#	Parameter name	Parameter description
1	Year	Year of the newly set date.
2	Month	Month of the newly set date.
3	Day	Day of the newly set date.

Example (Macro Steps):

1 <#> <#> This macro will set system date Feb 1, 2004

2 **Macro execution: ONLY COMMANDS**

3 **Set system DATE** Year=2004, Month=2, Day=1

Example (Plain Text):

```
<#> This macro will set system date Feb 1, 2004
<#>
<cmds>
<setsysdate>(2004,2,1)
```

Shutdown - < shutdown >() ... [Pro]

Shutdown
<shutdown>("Mode",Force)
Available in: Professional edition

This command shuts down/reboot computer or log off user.

#	Parameter name	Parameter description
1	Mode	Can be one of these values: "LOGOFF" "REBOOT" "SHUTDOWN" "POWEROFF"
2	Force	If 1, the shutdown will be forced even if there is an application that prevents system from shutdown. If 0, the shutdown will proceed only if all the running applications accepts system shutdown query.

Example (Macro Steps):

1 <#> <#> This macro will reboot computer

2 **Macro execution: ONLY COMMANDS**

3 Shutdown "Restart", Force="No"

Example (Plain Text):

```
<#> This macro will reboot computer
<#>
<cmds>
<shutdown>("REBOOT",0)
```

Registry CREATE KEY - < reg_createkey >() ... [Pro]

Registry CREATE KEY
<reg_createkey>("Registry key")
Available in: Professional edition

This command creates a registry key. There are these registry roots supported:
HKEY_CLASSES_ROOT
HKEY_CURRENT_CONFIG
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS
HKEY_PERFORMANCE_DATA
HKEY_DYN_DATA

#	Parameter name	Parameter description
1	Registry key	Full path of registry key to be created. Example: "HKEY_CURRENT_USER\Software\MyCompany\NewKey"

Example (Macro Steps):

1 <#> <#> This command create new 'TestKey key

2 <#> <#> in 'HKEY_CURRENT_USER\Software'

3 **Macro execution: ONLY COMMANDS**

4 Registry CREATE KEY "HKEY_CURRENT_USER\Software\TestKey"

Example (Plain Text):

```
<#> This command create new 'TestKey key
<#> in 'HKEY_CURRENT_USER\Software'
<cmds>
<reg_createkey>("HKEY_CURRENT_USER\Software\TestKey")
```

Registry DELETE KEY - < reg_deletekey >() ... [Pro]

Registry DELETE KEY
<reg_deletekey>("Registry key")
Available in: Professional edition

This command deletes specified registry key. There are these registry roots supported:
HKEY_CLASSES_ROOT
HKEY_CURRENT_CONFIG
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS
HKEY_PERFORMANCE_DATA
HKEY_DYN_DATA

#	Parameter name	Parameter description
1	Registry key	Full path of registry key to be deleted. Example: HKEY_CURRENT_USER\Software\MyCompany\NewKey

Example (Macro Steps):

1 <#> <#> This command deletes 'TestKey' key

2 <#> <#> from 'HKEY_CURRENT_USER\Software'

3 **Macro execution: ONLY COMMANDS**

4 Registry DELETE KEY "HKEY_CURRENT_USER\Software\TestKey"

Example (Plain Text):

```
<#> This command deletes 'TestKey' key
<#> from 'HKEY_CURRENT_USER\Software'
<cmds>
<reg_deletekey>("HKEY_CURRENT_USER\Software\TestKey")
```

Registry DELETE VALUE - < reg_deletevalue >() ... [Pro]

Registry DELETE VALUE
<reg_deletevalue>("Registry key","Value")
Available in: Professional edition

This command deletes defined registry value. There are these registry roots supported:
HKEY_CLASSES_ROOT
HKEY_CURRENT_CONFIG
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS
HKEY_PERFORMANCE_DATA
HKEY_DYN_DATA

#	Parameter name	Parameter description
1	Registry key	Full path of registry key to be deleted. Example: HKEY_CURRENT_USER\Software\MyCompany\NewKey
2	Value	Name of the value. Example: ProductName.

Example (Macro Steps):

1 <#> <#> This command retrieves ProductName value

2 <#> <#> of 'HKEY_CURRENT_USER\Software\TestKey'

3 **Macro execution: ONLY COMMANDS**

4 Registry DELETE VALUE Registry key=HKEY_CURRENT_USER\Software\TestKey, Value=ProductName

Example (Plain Text):

```
<#> This command retrieves ProductName value
<#> of 'HKEY_CURRENT_USER\Software\TestKey'
<cmds>
<reg_deletevalue>("HKEY_CURRENT_USER\Software\TestKey","ProductName")
```

Registry ENUMERATE SUBKEYS - < reg_enumsubkeys >() ... [Pro]

Registry ENUMERATE SUBKEYS
<reg_enumsubkeys>("Registry key","Variable array for enumerated items","Variable array size")
Available in: Professional edition

This command enumerates all subkeys of the given key. The result (array of subkeys names) is saved in variable. There are these registry roots supported:
HKEY_CLASSES_ROOT
HKEY_CURRENT_CONFIG
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS
HKEY_PERFORMANCE_DATA
HKEY_DYN_DATA

#	Parameter name	Parameter description
1	Registry key	Full path of registry key to be deleted. Example: HKEY_CURRENT_USER\Software\MyCompany\NewKey
2	Variable array for enumerated items	Variable (array) that receives subkeys found.
3	Variable array size	Variable that receives number of subkeys found.

Example (Macro Steps):

1 <#> <#> This example shows how to enumerate registry keys

2 <#> <#>

3 **Macro execution: ONLY COMMANDS**

4 **Registry ENUMERATE SUBKEYS** Registry key=HKEY_CURRENT_USER\Software, Variable array for enumerated items=vSubKey, Variable array size=vNumOfSubkeys

5 **Message SHOW** "" : "There are %vNumOfSubkeys% registry subkeys of 'HKEY_CURRENT_USER\Software' registry key. Showing first three: %vSubKey[0]% %vSubKey[1]% %vSubKey[2]% " (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

Example (Plain Text):

```
<#> This example shows how to enumerate registry keys
<#>
<cmds>
<reg_enumsubkeys>("HKEY_CURRENT_USER\Software","vSubKey","vNumOfSubkeys")
<msg>(-100,-100,"There are %vNumOfSubkeys% registry subkeys of 'HKEY_CURRENT_USER\Software' registry key.
Showing first three:
%vSubKey[0]%
%vSubKey[1]%
%vSubKey[2]%
","Message",1)
```

Registry ENUMERATE VALUES - < reg_enumvalues >() ... [Pro]

Registry ENUMERATE VALUES
<reg_enumvalues>("Registry key","Variable array for enumerated items","Variable array size")
Available in: Professional edition

This command enumerates all values of the given key. The result (array of subkeys names) is saved in variable. There are these registry roots supported:
HKEY_CLASSES_ROOT
HKEY_CURRENT_CONFIG
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS
HKEY_PERFORMANCE_DATA
HKEY_DYN_DATA

#	Parameter name	Parameter description
1	Registry key	Full path of registry key to be deleted. Example: HKEY_CURRENT_USER\Software\MyCompany\NewKey
2	Variable array for enumerated items	Variable (array) that receives values found.
3	Variable array size	Variable that receives number of values found.

Example (Macro Steps):

1 <#> <#> This example shows how to enumerate registry values

2 **Macro execution: ONLY COMMANDS**

3 **Registry ENUMERATE VALUES** Registry key=HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer, Variable array for enumerated items=vValue, Variable array size=vNumOfValues

4 **Message SHOW** "" : "There are %vNumOfValues% registry subkeys of 'HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer' registry key. Showing first three: %vValue[0]% %vValue[1]% %vValue[2]% " (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

Example (Plain Text):

```
<#> This example shows how to enumerate registry values
<#>
<cmds>
<reg_enumvalues>("HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer","vValue","vNumOfValues")
<msg>(-100,-100,"There are %vNumOfValues% registry subkeys of 'HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer' registry key. Showing first three:

%vValue[0]%
%vValue[1]%
%vValue[2]%
","Message",1)
```

Registry GET VALUE - < reg_getvalue >() ... [Pro]

Registry GET VALUE
<reg_getvalue>("Registry key","Value name","Variable to save data")
Available in: Professional edition

This command retrieves defined registry value. There are these registry roots supported:
HKEY_CLASSES_ROOT
HKEY_CURRENT_CONFIG
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS
HKEY_PERFORMANCE_DATA
HKEY_DYN_DATA

#	Parameter name	Parameter description
1	Registry key	Full path of registry key to be deleted. Example: HKEY_CURRENT_USER\Software\MyCompany\NewKey
2	Value name	Name of the value. Example: ProductName.
3	Variable to save data	Variable that receives value retrieved.

Example (Macro Steps):

1 <#> <#> This command retrieves ProductName value

2 <#> <#> of 'HKEY_CURRENT_USER\Software\TestKey'

3 **Macro execution: ONLY COMMANDS**

4 Registry GET VALUE "v" = value of "HKEY_CURRENT_USER\Software\TestKey\ProductName"

5 **Message SHOW** "" : "The retrieved value: %v%" (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

Example (Plain Text):

```
<#> This command retrieves ProductName value
<#> of 'HKEY_CURRENT_USER\Software\TestKey'
<cmds>
<reg_getvalue>("HKEY_CURRENT_USER\Software\TestKey","ProductName","v")
<msg>(-100,-100,"The retrieved value: %v%","Message",1)
```

Registry SET VALUE - < reg_setvalue >() ... [Pro]

Registry SET VALUE
<reg_setvalue>("Registry key","Value name","Value","Type")
Available in: Professional edition

This command sets defined registry value. There are these registry roots supported:
HKEY_CLASSES_ROOT
HKEY_CURRENT_CONFIG
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS
HKEY_PERFORMANCE_DATA
HKEY_DYN_DATA

#	Parameter name	Parameter description
1	Registry key	Full path of registry key to be deleted. Example: HKEY_CURRENT_USER\Software\MyCompany\NewKey
2	Value name	Name of the value. Example: ProductName.
3	Value	Value. Example: Macro Toolworks.
4	Type	0 - String value. 1 - Numeric value.

Example (Macro Steps):

1 <#> <#> This command sets ProductName value

2 <#> <#> of 'HKEY_CURRENT_USER\Software\TestKey'

3 <#> <#> to Something

4 **Macro execution: ONLY COMMANDS**

5 Registry SET VALUE "HKEY_CURRENT_USER\Software\TestKey\ProductName"" = "Something"

Example (Plain Text):

```
<#> This command sets ProductName value
<#> of 'HKEY_CURRENT_USER\Software\TestKey'
<#> to Something
<cmds>
<reg_setvalue>("HKEY_CURRENT_USER\Software\TestKey","ProductName","Something","0")
```

Printer SET DEFAULT - < printer_setdefault >() ... [Pro]

Printer SET DEFAULT
<printer_setdefault>("Printer name")
Available in: Professional edition

Sets specified printer to be the default one.

#	Parameter name	Parameter description
1	Printer name	Name of the printer to be set as default.

Example (Macro Steps):

1 <#> <#> This macro changes default printer

2 **Macro execution: ONLY COMMANDS**

3 Printer SET DEFAULT "PRN1"

Example (Plain Text):

```
<#> This macro changes default printer
<#>
<cmds>
<printer_setdefault>("PRN1")
```

Speakers VOLUME - < multimedia >() ... [Pro]

Speakers VOLUME
<multimedia>(Unused,Volume)
Available in: Professional edition

This command controls speakers.

#	Parameter name	Parameter description
1	Unused	Must be VOLUME.
2	Volume	One of these values: MUTE UNMUTE TOGGLEMUTE UP DOWN

Example (Macro Steps):

1 <#> <#> This macro will mute speakers

2 **Macro execution: ONLY COMMANDS**

3 Speakers VOLUME "Mute"

Example (Plain Text):

```
<#> This macro will mute speakers
<#>
<cmds>
<multimedia>(VOLUME,MUTE)
```

Process KILL - < process_kill >() ... [Pro]

Process KILL
<process_kill>(File)
Available in: Professional edition

This command kills defined process. Killing process is different from closing application (window) using "winclose" command. Killing process may result in loss of unsaved data in the process.

#	Parameter name	Parameter description
1	File	Name of the executable file (example: notepad.exe) to kill or process identification number (example: 6712).

Example (Macro Steps):

```
1    <#> <#> This example kills Notepad
2    <#> <#>
3    Macro execution: ONLY COMMANDS
4    Process KILL "notepad.exe"
```

Example (Plain Text):

```
<#> This example kills Notepad
<#>
<cmds>
<process_kill>(notepad.exe)
```

Screenasaver ENABLE - < scrsaver_enable > ... [Pro]

Screenasaver ENABLE
<scrsaver_enable>
Available in: Professional edition

This command enables screensaver.

Example (Macro Steps):

1 **<#> <#> This example enables screensaver**

2 **Screenasaver ENABLE**

Example (Plain Text):

<#> This example enables screensaver
<scrsaver_enable>

Screenasaver DISABLE - < scrsaver_disable > ... [Pro]

Screenasaver DISABLE
<scrsaver_disable>
Available in: Professional edition

This command disables screensaver.

Example (Macro Steps):

1 **<#> <#> This example disables screensaver**

2 **Screenasaver DISABLE**

Example (Plain Text):

<#> This example disables screensaver
<scrsaver_disable>

Process ENUMERATE - < process_enum >() ... [Pro]

Process ENUMERATE
<process_enum>("Variable array for names","Variable array size","Variable array for process Id's")
Available in: Professional edition

This command enumerates all running processes.

#	Parameter name	Parameter description
1	Variable array for names	Variable (array) that receives names of running processes.
2	Variable array size	Variable that receives number of process.
3	Variable array for process Id's	Variable (array) that receives Ids of running processes.

Example (Macro Steps):

```
1   <#> <#> This example enumerates running processes and lists them in Notepad
2   <#> <#>
3   Macro execution: ONLY COMMANDS
4   Process ENUMERATE Variable array for names=vProcess, Variable array size=vNumOfProcesses
5   Run APPLICATION "notepad.exe" (other parameters: Parameters = , Folder path = , Window state = Normal).
    Macro execution waits for application to finish up to "0" seconds.
6   WAIT FOR Object = "WIN", Event = "ACT", Parameter = "Notepad", Timeout (seconds) = "5", Exact = "0"
7   IF WINDOW "Notepad" Is Active (Match=Partial)
8       Macro execution: KEYS / FREE TEXT + COMMANDS
9       These processes are running #. [ name ;30 [ \Device\HarddiskVolume3\Program Files
        (x86)\iPass\Open Mobile\omsi\iPlatformService.exe ; 1772 ] Id ]:
10      Keyboard Insert NEW LINE
11      Keyboard Insert NEW LINE
12      Macro execution: ONLY COMMANDS
13      Loop BEGIN Repeat = vNumOfProcesses
14          Variable INSERT to active application "%_vLoopCounter% [ %vProcess[_vLoopCounter0]% ;
            %vProcessIds[_vLoopCounter0]% ]", Text insertion method="Default (as defined in settings)"
15          Macro execution: KEYS / FREE TEXT + COMMANDS
16          Keyboard Insert NEW LINE
17          Macro execution: ONLY COMMANDS
18      Loop END
19  ENDIF
```

Example (Plain Text):

```
<#> This example enumerates running processes and lists them in Notepad
<#>
<cmds>
<process_enum>("vProcess","vNumOfProcesses","vProcessIds")

<execappex>("notepad.exe","","",0,0)
<waitfor>("WIN","ACT","Notepad",5,0)
<if_win>("Notepad","ACT",0)
<keys>These processes are running #. [ name ;30 [ \Device\HarddiskVolume3\Program Files (x86)\iPass\Open
Mobile\omsi\iPlatformService.exe ; 1772 ] Id ]:<newline><#>
<newline><#>
<cmds>
<begloop>(vNumOfProcesses)
<varout>("%_vLoopCounter% [ %vProcess[_vLoopCounter0]% ; %vProcessIds[_vLoopCounter0]% ]","0")<#>
<keys><#>
<newline><#>
```

<cmds>
<endloop>

<endif>

WINDOWS SERVICE - < winsvc >() ... [Pro]

WINDOWS SERVICE
<winsvc>("Service Name","Command")
Available in: Professional edition

The command changes state of a Windows service.
Note: This command requires the program to run with "Administrator privilages"

#	Parameter name	Parameter description
1	Service Name	The Windows service name
2	Command	Command to change the Windows service state: "START" - starts the service. "STOP" - stops the service.

Example (Macro Steps):

```
1    <#> <#>This macro shows how to use "if_winsvc" and "winsvc" commands
2    Macro execution: ONLY COMMANDS
3    IF WINDOWS SERVICE "WebClient" is "NOT running" then execute following steps
4        WINDOWS SERVICE "WebClient", Command="Start"
5    ENDIF
```

Example (Plain Text):

```
<#>This macro shows how to use "if_winsvc" and "winsvc" commands
<cmds>
<if_winsvc>("WebClient","IS_NOT_RUNNING")
  <winsvc>("WebClient","START")
<endif>
```

Text & Variable Manipulation

SET - < varset >() ... [Pro]

Variable SET
<varset>("Variable","Message text")
Available in: Professional edition

Declares variable and sets its value.

#	Parameter name	Parameter description
1	Variable	Defines variable and its value in the form "variable=VALUE". Variable can be any string (different from system variables). The value can be either constant string or other variable or empty string. There are four basic behaviors of the command: 1. VALUE is supplied. In such case the variable is assigned the VALUE. For example, "vNumber=10" or "vName=John". 2. List of possible values is supplied in VALUE. Each single item in the list is separated by "_OR_" ("|" delimiter character is depricated since version 8.1.0). For example, "vNames=Jane_OR_Paul_OR_Chris_OR_Michael". 3. VALUE is an empty string ("vName="). In such the case, a dialog window with the MessageTitle appears and prompts user to enter the variable value. Note: To assign an empty string to the variable use this construction: "vEmptyStr=_vStrEmpty". 4. VALUE is equal to "YES/NO". In such the case, a "yes/no" message box withMessageTitle appears. If user selects "yes" button the "YES" value is assigned to the variable. If the user selects "no" button the "NO" value is assigned. For example, "vDoYouWant=YES/NO". In cases #2 and #3 above, if user clicks "Cancel" button the _vCanceled system variable is set to 1 (otherwise it is 0) to allow macro designer to distinguish between situations when OK or Cancel button was clicked. Instead of using VARIABLE_NAME=VALUE syntax, also VARIABLE_NAME=!VALUE can be used. In this case, the VALUE is assigned to the variable and the rules #2 - #4 above are not applied (hard assignement).
2	Message text	A title of the window that appears in cases 1 and 3 above.

Example (Macro Steps):

```
1   <#> <#> This macro demonstrates command
2   Macro execution: ONLY COMMANDS
3   Variable SET "vName=", Message text="Insert Your Name"
4   IF STRING _vCanceled==1
5       Macro EXIT
6   ENDIF
7   Variable SET "vShow=YES/NO", Message text="Do you want to see your name ?"
8   IF STRING vShow==YES
9       Message SHOW "" : "vName" (other parameters: x = 100, y = 100, Window title = Your name is:, Buttons
        = OK, Timeout (seconds) = , Always on top = ).
10  ENDIF
11  Variable SET "vName=Jane_OR_Paul_OR_Michael_OR_Chris", Message text="Who is my best friend ?"
12  IF STRING _vCanceled==1
13      Macro EXIT
14  ENDIF
15  IF STRING vName==Jane
16      Message SHOW "" : "Yes!" (other parameters: x = 100, y = 100, Window title = Message, Buttons = OK,
        Timeout (seconds) = , Always on top = ).
17  ELSE activate
18      Message SHOW "" : "No!" (other parameters: x = 100, y = 100, Window title = Message, Buttons = OK,
        Timeout (seconds) = , Always on top = ).
19  ENDIF
```

Example (Plain Text):

```
<#> This macro demonstrates <varset> command
<#>
<cmds>

<varset>("vName=","Insert Your Name")
<if_str>("_vCanceled==1") <exitmacro> <endif>

<varset>("vShow=YES/NO","Do you want to see your name ?")

<if_str>("vShow==YES")
   <msg>(100,100,"vName","Your name is:",1)
<endif>

<varset>("vName=Jane_OR_Paul_OR_Michael_OR_Chris","Who is my best friend ?")
<if_str>("_vCanceled==1") <exitmacro> <endif>
```

```
<if_str>("vName==Jane")
   <msg>(100,100,"Yes!","Message",1)
<else>
   <msg>(100,100,"No!","Message",1)
<endif>
```

INSERT to active application - < varout >() ... [Pro]

Variable INSERT to active application
<varout>("Variable",Text insertion method)
Available in: Professional edition

Sends (outputs) variable's value to the active window as a sequence of keystrokes or through the clipboard.

#	Parameter name	Parameter description
1	Variable	Variable to send out.
2	Text insertion method	If 0, the variable content is send to the active window the way it is defined in program settings ("Options" dialog box - click "Tools/Options" menu command). If 1, the variable value is send to the active window through the clipboard. If 2, the variable value is send to the active window as a sequence of keyboard keystrokes.

Example (Macro Steps):

```
1   <#> <#> This macro demonstrates use of command
2   <#> <#> The macro activates Notepad and writes a text in it.
3   Macro execution: ONLY COMMANDS
4   IF WINDOW "Notepad" Is Open (Match=Partial)
5       Window ACTIVATE bring "Notepad" window to top (other parameters: Match = Partial, Window state =
        Normal, %p4_name = no)
6       WAIT FOR Object = "WIN", Event = "ACT", Parameter = "Notepad", Timeout (seconds) = "5", Exact = "0"
7       IF WINDOW "Notepad" Is Active (Match=Partial)
8           Variable SET "vText=This is Notepad", Message text=""
9           Variable INSERT to active application "vText", Text insertion method="Default (as defined in
            settings)"
10      ENDIF
11  ELSE activate
12      Message SHOW "" : "Notepad is not running." (other parameters: x = 100, y = 100, Window title =
        Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
13  ENDIF
```

Example (Plain Text):

```
<#> This macro demonstrates use of <varout> command
<#> The macro activates Notepad and writes a text in it.
<#>
<cmds>
```

```
<if_win>("Notepad","OPEN",0)
   <actwin>("Notepad",0,0,"no")
   <waitfor>("WIN","ACT","Notepad",5,0)
   <if_win>("Notepad","ACT",0)
      <varset>("vText=This is Notepad","")
      <varout>("vText",0)
   <endif>
<else>
   <msg>(100,100,"Notepad is not running.","Message",1)
<endif>)
```

SAVE - < var_save >() ... [Pro]

Variable SAVE
<var_save>("Variable","File")
Available in: Professional edition

The command saves (persists) variable value to a file. The variable content can be later loaded again using command. The same file can be used for multiple (unlimited number) variables because each variable has its own section within the file.

#	Parameter name	Parameter description
1	Variable	Variable to save value to file.
2	File	The file to save variable value to.

Example (Macro Steps):

1 <#> <#> This macro demonstrates use of and commands

2 **Macro execution: ONLY COMMANDS**

3 **Variable SET** "vName=John", Message text=""

4 **Variable SAVE** "vName" to file "c:_myvars.dat

5 **Variable SET** "vName=Michael", Message text=""

6 **Variable LOAD** from file "c:_myvars.dat" to "vName"

7 **Message SHOW** "" : "vName" (other parameters: x = 100, y = 100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

Example (Plain Text):

```
<#> This macro demonstrates use of <var_load> and <var_save> commands
<#>
<cmds>
<varset>("vName=John","")
<var_save>("vName","c:\_myvars.dat")
<varset>("vName=Michael","")
<var_load>("vName","c:\_myvars.dat")
<msg>(100,100,"vName","Message",1)
```

LOAD - < var_load >() ... [Pro]

Variable LOAD
<var_load>("Variable","File")
Available in: Professional edition

Loads variable value from a file. (The value was previously saved using command.)

#	Parameter name	Parameter description
1	Variable	Variable to load value from file.
2	File	The file created by command that contains variable value.

Example (Macro Steps):

1 <#> <#> This macro demonstrates use of and commands

2 **Macro execution: ONLY COMMANDS**

3 **Variable SET** "vName=John", Message text=""

4 **Variable SAVE** "vName" to file "c:_myvars.dat

5 **Variable SET** "vName=Michael", Message text=""

6 **Variable LOAD** from file "c:_myvars.dat" to "vName"

7 **Message SHOW** "" : "vName" (other parameters: x = 100, y = 100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

Example (Plain Text):

```
<#> This macro demonstrates use of <var_load> and <var_save> commands
<#>
<cmds>
<varset>("vName=John","")
<var_save>("vName","c:\_myvars.dat")
<varset>("vName=Michael","")
<var_load>("vName","c:\_myvars.dat")
<msg>(100,100,"vName","Message",1)
```

PARSE - < var_parse >() ... [Pro]

Variable PARSE
<var_parse>("Input text/variable","Delimiter","Trim character",Variable array for enumerated items,Variable array size)
Available in: Professional edition

This command parses input text (variable value). Parts of the input text are expected to be delimited by characters (space, comma, new line, etc.) defined as a command parameter. The parsed parts of the input text are saved in parameter variable (array). In addition, each parsed part can be trimmed using characters defined.

#	Parameter name	Parameter description
1	Input text/variable	Text (or variable containing text) to be parsed.
2	Delimiter	Characters used to delimit input text (for example, dot, comma, space new line, etc.).
3	Trim character	Characters used to trim each parsed part of the input text.
4	Variable array for enumerated items	Variable (array) that receives each input text piece parsed.
5	Variable array size	Variable that receives the number of pieces.

Example (Macro Steps):

1 <#> <#> This example shows how to use command

2 <#> <#> The input text consisting from names delimited by comma

3 <#> <#> will be parsed and each name will be shown in message box.

4 **Macro execution: ONLY COMMANDS**

5 **Variable PARSE** "Peter, Paul, John, Jim " to variable array "vName" (Delimiter = ,, Trim character = , Variable array for enumerated items = vName, Variable array size = vNumOfNames)

6 **Loop BEGIN** Repeat = vNumOfNames

7 **Message SHOW** "" : "vName[_vLoopCounter0]" (other parameters: x = 100, y = 100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

8 **Loop END**

Example (Plain Text):

```
<#> This example shows how to use <var_parse> command
<#> The input text consisting from names delimited by comma
<#> will be parsed and each name will be shown in message box.
<cmds>
<var_parse>("Peter,  Paul,  John,   Jim  ",",","," ",vName,vNumOfNames)
<begloop>(vNumOfNames)
  <msg>(100,100,"vName[_vLoopCounter0]","Message",1)
<endloop>
```

OPERATION - < var_oper >() ... [Pro]

Variable OPERATION

<var_oper>(Variable for result,"Input text/variable",Operation,"Parameter 1","Parameter 2","Parameter 3")
Available in: Professional edition

Performs selected operation on input variable (or constant string/value) and saves the result to other variable.

#	Parameter name	Parameter description
1	Variable for result	Variable that receives result of the operation.
2	Input text/variable	Either variable or constant text (string, numerical value) to perform operation on.
3	Operation	One of these operations: CALC_EXPRESSION - calculates arithmetical expression defined by input. "Parameter 1" tells how many numbers will follow after decimal point. RANDOM_NUMBER - generates random number from 0 to "Parameter 1" range. SELECT_FILE - shows "Open File" dialog and saves full path of the selected file to "Variable for result". If user clicks "Cancel" button the _vCanceled system variable is set to 1 (otherwise it is 0) to allow macro designer to distinguish between situations when OK or Cancel button was clicked. SELECT_FOLDER - shows "Select Folder" dialog and saves full path of the selected folder to "Variable for result". If user clicks "Cancel" button the _vCanceled system variable is set to 1 (otherwise it is 0) to allow macro designer to distinguish between situations when OK or Cancel button was clicked. GET_TEXT_FROM_CLIPBOARD - copies text from clipboard to "Variable for result". STR_APPEND - appends the Input to the content of "Variable for result". STR_LEFT - characters from begin of the Input (number of characters is given by "Parameter 1") are copied to "Variable for result". STR_RIGHT - characters from end of the Input (number of characters is given by "Parameter 1") are copied to "Variable for result". STR_MID - characters beginning at "Parameter 1" position (number of characters is given by "Parameter 2") are copied to "Variable for result". STR_TRIMLEFT - characters specified in "Parameter 1" are deleted from begin of the Input. STR_TRIMRIGHT - characters specified in "Parameter 1" are deleted from end of the Input. STR_INSERT - string defined by "Parameter 2" is inserted to Input string to position "Parameter 1". STR_REPLACE - In the Input, original string "Parameter 1" is replaced by "Parameter 2". STR_DELETE - in the Input, deletes "Parameter 2" (number of characters to delete) characters starting at position "Parameter 1". STR_FIND - finds first occurrence of the "Parameter 1" string in the Input. The search is started from "Parameter 2" position. If not found then the "Variable for result"

Example (Macro Steps):

```
1   <#> <#> This example converts text in clipboard to upper-case
2   Macro execution: ONLY COMMANDS
3   Variable OPERATION "GET_TEXT_FROM_CLIPBOARD" (Variable for result = vClipboardText, Input
    text/variable = , Parameter 1 = , Parameter 2 = , Parameter 3 = 0)
4   IF STRING vClipboardText!=_vStrEmpty
5       Variable OPERATION "STR_UPPER" (Variable for result = vClipboardText, Input text/variable =
        %vClipboardText%, Parameter 1 = , Parameter 2 = , Parameter 3 = 0)
6       Clipboard COPY "vClipboardText"
7       Message SHOW "" : "Text in clipboard is converted to uppercase." (other parameters: x = -100, y = -100,
        Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
8   ENDIF
```

Example (Plain Text):

```
<#> This example converts text in clipboard to upper-case
<#>
<cmds>
<var_oper>(vClipboardText,"",GET_TEXT_FROM_CLIPBOARD,"","", "0")
<if_str>("vClipboardText!=_vStrEmpty")

  <var_oper>(vClipboardText,"%vClipboardText%",STR_UPPER,"","", "0")
  <clpput>("vClipboardText")
  <msg>(-100,-100,"Text in clipboard is converted to uppercase.","Message",1)

<endif>
```

Regular Expression Find - < regex_find >() ... [Pro]

Variable Regular Expression Find

<regex_find>("Input text/variable","Pattern","Start index",Variable for match,Variable for index,Variable for size)
Available in: Professional edition

This command searches in the input text for a matching regular expression pattern. The command searches from the position passed as "Start index". If a match is found then the matching substring, index where it is located in the input text, and its length is returned.

#	Parameter name	Parameter description
1	Input text/variable	Input text.
2	Pattern	Regular expression pattern.
3	Start index	Index (position) in the text where to start searching. 0 means that the input text is searched from beginning.
4	Variable for match	Name of the variable that will receive substring that matches the regular expression pattern. If no match is found then this variable will contain empty string (equal to %_vStrEmpty%).
5	Variable for index	Name of the variable that will receive index (position) in the input text where the match was found. If no match is found then this variable will contain -1.
6	Variable for size	Name of the variable that will receive the length of the matching substring found. If no match is found then this variable will contain 0.

Example (Macro Steps):

```
1   <#> <#>This example shows how to find an e-mail addresses in input text
2   Macro execution: ONLY COMMANDS
3   Variable SET "vSearchFrom=0", Message text=""
4   Repeat steps UNTIL "1" (Counter variable initial value = "", Counter loop increment = "")
5       Variable Regular Expression Find Pattern "\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}" in "This is
        a text that contains e-mail addresses such as john.a.smith@comp-comp-company.com or
        FredieX@dot.dot.TV or other...." (Start index=%vSearchFrom%, Variable for match = vM, Variable for index
        = vI, Variable for size = vS)
6       IF %vM%==%_vStrEmpty%
7           Message SHOW "Information" : "No more e-mail addresses found." (other parameters: x = -100, y =
            -100, Window title = , Buttons = OK, Timeout (seconds) = 0, Always on top = No).
8           Repeat steps BREAK
9       ELSE activate
10          Message SHOW "Information" : "E-mail found: %vM% Position in text: %vI% Length: %vS%" (other
            parameters: x = -100, y = -100, Window title = , Buttons = OK, Timeout (seconds) = 0, Always on top =
            No).
11          Variable SET "%vSearchFrom%=EXPR(%vI%+%vS%)", Message text=""
12      ENDIF
13  Repeat steps END
```

Example (Plain Text):

```
<#>This example shows how to find an e-mail addresses in input text
<cmds>>
<varset>("vSearchFrom=0","")
<for>("","1","")
<regex_find>("This is a text that contains e-mail addresses such as john.a.smith@comp-comp-company.com or
FredieX@dot.dot.TV or other....","\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}","%vSearchFrom%",vM,vI,vS)
<if>("%vM%==%_vStrEmpty%")
  <msg>(-100,-100,"No more e-mail addresses found.","",1,0,0,0)
  <for_break>
<else>
  <msg>(-100,-100,"E-mail found: %vM%
Position in text: %vI%
Length: %vS%","",1,0,0,0)
  <varset>("%vSearchFrom%=EXPR(%vI%+%vS%)","")
<endif>
<for_end>
```

ENCRYPT/DECRYPT - < data_crypt >() ... [Pro]

Variable ENCRYPT/DECRYPT
<data_crypt>("Input",Variable for result,"Password",ENCRYPT/DECRYPT)
Available in: Professional edition

This command encodes or decodes input text using provided password. The encoded/decoded text is put into variable. The variable can be used in other macro commands. If input text cannot be decoded due to a wrong password then empty string is saved in the variable.

#	Parameter name	Parameter description
1	Input	The input text to be encoded/decoded.
2	Variable for result	Name of the variable the encoded/decoded text is saved in.
3	Password	Password used to encode/decode the input text.
4	ENCRYPT/DECRYPT	ENCRYPT or DECRYPT

Example (Macro Steps):

1 <#> <#>This example shows how to use 'data_crypt' command

2 **Macro execution: ONLY COMMANDS**

3 **Variable SET** "vPwd1=", Message text="Enter password to encrypt text"

4 **Variable ENCRYPT/DECRYPT** Input = "Original input text to be encrypted", Variable for result = "vCT", Password = "%vPwd1%", ENCRYPT/DECRYPT = "ENCRYPT"

5 **Message SHOW** "Information" : "%vCT%" (other parameters: x = -100, y = -100, Window title = Encrypted text looks like this, Buttons = OK, Timeout (seconds) = 0, Always on top = No).

6 **Variable SET** "vPwd2=", Message text="Enter password to decrypt text"

7 **Variable ENCRYPT/DECRYPT** Input = "%vCT%", Variable for result = "vCT2", Password = "%vPwd2%", ENCRYPT/DECRYPT = "DECRYPT"

8 **Message SHOW** "Information" : "%vCT2%" (other parameters: x = -100, y = -100, Window title = If you provided correct decrypt password you see the original text, Buttons = OK, Timeout (seconds) = 0, Always on top = No).

Example (Plain Text):

```
<#>This example shows how to use 'data_crypt' command
<cmds>
<varset>("vPwd1=","Enter password to encrypt text")
<data_crypt>("Original input text to be encrypted",vCT,"%vPwd1%",ENCRYPT)
<msg>(-100,-100,"%vCT%","Encrypted text looks like this",1,0,0,0)

<varset>("vPwd2=","Enter password to decrypt text")
<data_crypt>("%vCT%",vCT2,"%vPwd2%",DECRYPT)
<msg>(-100,-100,"%vCT2%","If you provided correct decrypt password you see the original text",1,0,0,0)
```

PARSE - < text_parse >() ... [Pro]

Variable PARSE
<text_parse>("Input text/variable","Pattern","Variable for matching elements","Variable for number of matching elements","Additional options")
Available in: Professional edition

This command parses input text to multiple pieces based on the pattern provided. The pattern consists of asterisks (*) and delimter substrings. The parses extracts the parts of the text that correspond to asterisks.
Example:
Input text:
Customer info:
Name: John Smith
Phone: +001 564123123 Email: jsmith@email.com
Pattern: Name:*Phone:*Email:*
Data extracted by parser as result: John Smith, +001 564123123, jsmith@email.com

#	Parameter name	Parameter description
1	Input text/variable	Input text or variable (such as %vInputText%)
2	Pattern	Pattern or variable containing pattern text
3	Variable for matching elements	Name of the array variable that will receive the parsed out pieces of the input text
4	Variable for number of matching elements	Name of the variable that contains number of items in the array variable
5	Additional options	Parse options (options are case sensitive): -nc - Not case sensitive when matching delimiter substrings. -tl - trim left. Spaces, tabs, new lines are removed from the beggining of the parsed out pieces. -tr - trim right. Spaces, tabs, new lines are removed from the end of the parsed out pieces. -rp - Repeat pattern

Example (Macro Steps):

1 <#> <#> This example demonstrates how to use "text parse" command

2 **Variable SET** "vInput=Customers info: Name: John Smith Phone: +001 564123123 Email: jsmith@email.com Name: Jack Back Phone: +002 779835 Email: jb@comp.com", Message text=""

3 **Variable PARSE** Input text/variable = %vInput%, Pattern = name:*phone:*email:*, Variable for matching elements = vItems, Variable for number of matching elements = vItemsSize, Additional options = -nc -tl -tr -rp

4 **Message SHOW** "Information" : "Input text is: %vInput%" (other parameters: x = -100, y = -100, Window title = , Buttons = OK, Timeout (seconds) = 0, Always on top = No).

5 **Message SHOW** "Information" : "%vItemsSize% items retrieved: %vItems[0]% %vItems[1]% %vItems[2]% %vItems[3]% %vItems[4]% %vItems[5]% " (other parameters: x = -100, y = -100, Window title = , Buttons = OK, Timeout (seconds) = 0, Always on top = No).

Example (Plain Text):

```
<#> This example demonstrates how to use "text parse" command
<#>
```

```
<varset>("vInput=Customers info:
Name: John Smith
Phone: +001 564123123
Email: jsmith@email.com
Name: Jack Back
Phone: +002 779835
Email: jb@comp.com","")<#>
<text_parse>("%vInput%","name:*phone:*email:*","vItems","vItemsSize","-nc -tl -tr -rp","p6")<#>
<msg>(-100,-100,"Input text is:

%vInput%","",1,0,0,0)<#>
<msg>(-100,-100,"%vItemsSize% items retrieved:

%vItems[0]%
%vItems[1]%
%vItems[2]%
%vItems[3]%
%vItems[4]%
%vItems[5]%
","",1,0,0,0)
```

User Interaction

SOUND - < beep >() ... [Pro]

SOUND
<beep>("Sound file",Repeat)
Available in: Professional edition

This command plays user defined sound (.wav file) or just beeps.

#	Parameter name	Parameter description
1	Sound file	Full path to the file (.wav) to play. If empty then just a beep is played.
2	Repeat	How many times the sound should be repeated.

Example (Macro Steps):

1 <#> <#> This macro will play the sound file you select

2 **Macro execution: ONLY COMMANDS**

3 **Variable OPERATION** "SELECT_FILE" (Variable for result = vSoundFile, Input text/variable = , Parameter 1 = Select File, Parameter 2 = , Parameter 3 = 0)

4 **SOUND** play sound "%vSoundFile%" for "1" times

Example (Plain Text):

```
<#> This macro will play the sound file you select
<#>
<cmds>
<var_oper>(vSoundFile,"",SELECT_FILE,"Select File","", "0")
<beep>("%vSoundFile%",1)
```

Message SHOW - < msg >() ... [Free]

Message SHOW
<msg>(x,y,"Message text","Window title",Buttons,Timeout (seconds),Type of message,Always on top)
Available in: Free edition

Shows a message window on the screen.

#	Parameter name	Parameter description
1	x	X-coordinate of the message window position. If both X-coordinate and Y-coordinate is equal to -100 then the message window is centered on the screen.
2	y	Y-coordinate of the message window position.
3	Message text	Text that is displayed in the message window.
4	Window title	Title of the message window.
5	Buttons	Button(s) available in the message window. When user clicks the button then the message window is closed. Can be one of these values: 0 - No button to close the message window is available. The "msgoff" command must be used to close the message window. 1 - OK button is shown in the message window. 2 - Yes and No buttons are shown in the message window. If user clicks Yes button, the "_vMsgButton" system variable is set to "YES". Otherwise the "_vMsgButton" variable contains "NO:.
6	Timeout (seconds)	Timeout in seconds. If the message is still showing on the screen after this time period then it is automatically closed.
7	Type of message	Type of icon that will be displayed in the message window.
8	Always on top	Allows to keep the message window always on top of other windows so that it is visible to the user.

Example (Macro Steps):

```
1   <#> <#> This macro will show various use of message window
2   Macro execution: ONLY COMMANDS
3   Message SHOW "Information" : "This is an information message...." (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = 0, Always on top = ).
4   Message SHOW "Information" : "...and this message has 10 seconds timeout...." (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = 10, Always on top = ).
5   Message SHOW "Question" : "...and you can also answer questions...." (other parameters: x = -100, y = -100, Window title = Message, Buttons = Yes and No, Timeout (seconds) = 0, Always on top = ).
6   IF _vMsgButton==YES
7       Message SHOW "Information" : "You clicked YES!" (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = 0, Always on top = ).
8   ELSE activate
9       Message SHOW "Error" : "You clicked NO" (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = 0, Always on top = ).
10  ENDIF
11  Message SHOW "Information" : "And this message without a button will close if you press 'F10' key." (other parameters: x = -100, y = -100, Window title = Message, Buttons = None, Timeout (seconds) = 0, Always on top = ).
12  WAIT FOR Object = "KEY", Event = "PRESS", Parameter = "", Timeout (seconds) = "15", Exact = "0"
13  Message CLOSE
14  <#> <#>Message content is updated several times:
15  Message SHOW "Information" : "Message content A (Wait for 2 seconds)" (other parameters: x = -100, y = -100, Window title = Message A, Buttons = None, Timeout (seconds) = 0, Always on top = No).
16  WAIT wait "2000" ms (time is constant: "No")
17  Message SHOW "Information" : "Message content B (Wait for 2 seconds)" (other parameters: x = -100, y = -100, Window title = Message B, Buttons = None, Timeout (seconds) = 0, Always on top = No).
18  WAIT wait "2000" ms (time is constant: "No")
19  Message SHOW "Information" : "Message content C (Wait for 2 seconds)" (other parameters: x = -100, y = -100, Window title = Message C, Buttons = None, Timeout (seconds) = 0, Always on top = No).
20  WAIT wait "2000" ms (time is constant: "No")
21  Message CLOSE
```

Example (Plain Text):

```
<#> This macro will show various use of message window
<cmds>

<msg>(-100,-100,"This is an information message....","Message",1,0,0)
<msg>(-100,-100,"...and this message has 10 seconds timeout....","Message",1,10,0)
```

```
<msg>(-100,-100,"...and you can also answer questions....","Message",2,0,1)
<if>("_vMsgButton==YES")
   <msg>(-100,-100,"You clicked YES!","Message",1,0,0)
<else>
   <msg>(-100,-100,"You clicked NO","Message",1,0,2)
<endif>

<msg>(-100,-100,"And this message without a button will close if you press 'F10' key.","Message",0,0,0)
<waitfor>("KEY","PRESS","<F10>",15,0)
<msgoff>

<#>Message content is updated several times:
<msg>(-100,-100,"Message content A
(Wait for 2 seconds)","Message A",0,0,0,0)
<wx>(2000,0)
<msg>(-100,-100,"Message content B
(Wait for 2 seconds)","Message B",0,0,0,0)
<wx>(2000,0)
<msg>(-100,-100,"Message content C
(Wait for 2 seconds)","Message C",0,0,0,0)
<wx>(2000,0)
<msgoff>
```

Message CLOSE - < msgoff > ... [Free]

Message CLOSE
<msgoff>
Available in: Free edition

This command closes message window that was previously opened with no buttons to show. If the message window command is called subsequently before it is closed by "msgoff" then the message content is updated and message box window is resized (if necessary).

Example (Macro Steps):

1 <#> <#> This macro will display message window and closes it after 5 seconds

2 **Macro execution: ONLY COMMANDS**

3 **Message SHOW** "" : "Wait for 5 seconds..." (other parameters: x = -100, y = -100, Window title = Message, Buttons = None, Timeout (seconds) = , Always on top =).

4 **WAIT** wait "5000" ms (time is constant: "")

5 **Message CLOSE**

Example (Plain Text):

```
<#> This macro will display message window and closes it after 5 seconds
<#>
<cmds>
<msg>(-100,-100,"Wait for 5 seconds...","Message",0)
<wx>(5000)
<msgoff>
```

E-mail COMPOSE - < email >() ... [Pro]

E-mail COMPOSE
<email>("E-mail address",Window state)
Available in: Professional edition

Creates new e-mail message using default e-mail client.

#	Parameter name	Parameter description
1	E-mail address	E-mail address (e.g., support@mycompany.com). The address can be empty.
2	Window state	The state of the window: 0 - Normal 1 - Maximized 2 - Minimized

Example (Macro Steps):

1 <#> <#> This command creates new mail message

2 **Macro execution: ONLY COMMANDS**

3 **E-mail COMPOSE** create new e-mail using default mail client. E-mail address = , Window state = Normal

Example (Plain Text):

```
<#> This command creates new mail message
<#>
<cmds>
<email>("",0)
```

Net drive WINDOW connect - < netcondrivedlg > ... [Pro]

Net drive WINDOW connect
<netcondrivedlg>
Available in: Professional edition

This command opens a window that allows to connect (map) a network folder to a local drive letter.

Example (Macro Steps):

1 **<#> <#> This macro opens a window to map network folder to a drive letter**

2 **Net drive WINDOW connect**

Example (Plain Text):

```
<#> This macro opens a window to map network folder to a drive letter
<netcondrivedlg>
```

Net drive WINDOW disconnect - < netdiscondrivedlg > ... [Pro]

Net drive WINDOW disconnect
<netdiscondrivedlg>
Available in: Professional edition

This command opens a window that allows to disconnect (unmap) a network folder from a local drive letter.

Example (Macro Steps):

1 <#> <#> This macro opens a window to unmap network folder from a drive letter

2 **Net drive WINDOW disconnect**

Example (Plain Text):

```
<#> This macro opens a window to unmap network folder from a drive letter
<netdiscondrivedlg>
```

Form OPEN - < form_show >() ... [Pro]

Form OPEN
<form_show>("Form identifier","Window title","Icon file",Icon index,Width,Clear items from form on close,x,y,Show OK and Cancel buttons,Number of columns)
Available in: Professional edition

This command displays the form with fields previously added by "form_item" commands. If the the "Cancel" button is clicked or Esc key is pressed then the _vCanceled system variable is set to 1. Otherwise it is 0 indicating the user did not cancel the form.

#	Parameter name	Parameter description
1	Form identifier	An identifier of the form (for example, "FM1").
2	Window title	The title of the form window.
3	Icon file	The full path to the file with the icon to show.
4	Icon index	Index of the icon within the icon file.
5	Width	Width of the form window in pixels. If left empty, the default value is used.
6	Clear items from form on close	If set to 1, all form fields are removed after the form is closed. It is necessary to call "form_item" commands before the same form can be shown again by "form_show" command. If the option is 0 then the form fields remain attached to the form.
7	x	X-coordinate of the form position on the computer screen. If not supplied, the form is centered.
8	y	Y-coordinate of the form position on the computer screen. If not supplied, the form is centered.
9	Show OK and Cancel buttons	If this parameter is set to "1" then "OK" and "Cancel" buttons are automatically displayed in the form. If this parameter is set to "0" then "OK" and "Cancel" buttons are not showing in the form window and it contains only controls that are added by "form_item" command.
10	Number of columns	Number of columns. It is possible to arrange the fields into multiple columns.

Example (Macro Steps):

```
1  <#> <#> This example shows how to use form commands
2  Macro execution: ONLY COMMANDS
3  Form FIELD "This is a simple calculator for: + - * /" of type "Static text" (Default value=0, Variable to save field value=, Form identifier=FM1)
4  Form FIELD "" of type "Separator" (Default value=0, Variable to save field value=, Form identifier=FM1)
5  Form FIELD "Operand 1:" of type "Text edit" (Default value=0, Variable to save field value=vOper1, Form identifier=FM1)
6  Form FIELD "Operation:" of type "Drop down list" (Default value=Plus|Minus|Multiply|Divide, Variable to save field value=vOperation, Form identifier=FM1)
7  Form FIELD "Operand 2:" of type "Text edit" (Default value=0, Variable to save field value=vOper2, Form identifier=FM1)
8  Form FIELD "" of type "Separator" (Default value=0, Variable to save field value=, Form identifier=FM1)
9  Form FIELD "Continue ?" of type "Check box" (Default value=YES, Variable to save field value=vAgain, Form identifier=FM1)
10 Jump TARGET "lbl_Again"
11 Form OPEN "FM1", Window title="Simple Calculator"
12 IF STRING _vCanceled==1
13     Macro EXIT
14 ENDIF
15 IF STRING vOperation==Plus
16     Variable SET "vOper1=EXPR(%vOper1%+%vOper2%)", Message text=""
17 ENDIF
18 IF STRING vOperation==Minus
19     Variable SET "vOper1=EXPR(%vOper1%-%vOper2%)", Message text=""
20 ENDIF
21 IF STRING vOperation==Multiply
22     Variable SET "vOper1=EXPR(%vOper1%*%vOper2%)", Message text=""
23 ENDIF
24 IF STRING vOperation==Divide
25     Variable SET "vOper1=EXPR03(%vOper1%/%vOper2%)", Message text=""
26 ENDIF
27 Message SHOW "Information" : "vOper1" (other parameters: x = -100, y = -100, Window title = The result is:,
```

Example (Plain Text):

```
<#> This example shows how to use form commands
<cmds>
 <form_item>("FM1","This is a simple calculator for: + - * /","TEXT","0","",1)
<form_item>("FM1","","LINE","0","",1)
<form_item>("FM1","Operand 1:","EDIT","0","vOper1",1)
<form_item>("FM1","Operation:","LIST","Plus|Minus|Multiply|Divide","vOperation",1)
<form_item>("FM1","Operand 2:","EDIT","0","vOper2",1)
<form_item>("FM1","","LINE","0","",1)
<form_item>("FM1","Continue ?","CHECK","YES","vAgain",1)
<label>("lbl_Again")

<form_show>("FM1","Simple Calculator","calc.exe",0,500,0,,,1,1)
<if_str>("_vCanceled==1") <exitmacro> <endif>

<if_str>("vOperation==Plus")<#>
<varset>("vOper1=EXPR(%vOper1%+%vOper2%)","")
<endif>
<if_str>("vOperation==Minus")
   <varset>("vOper1=EXPR(%vOper1%-%vOper2%)","")
<endif>
<if_str>("vOperation==Multiply")
   <varset>("vOper1=EXPR(%vOper1%*%vOper2%)","")
<endif>
<if_str>("vOperation==Divide")
   <varset>("vOper1=EXPR03(%vOper1%/%vOper2%)","")
<endif>
<msg>(-100,-100,"vOper1","The result is:",1,,0)
<if_str>("vAgain==YES")
   <goto>("lbl_Again")
<endif>
```

Form FIELD - < form_item >() ... [Pro]

Form FIELD
<form_item>("Form identifier","Field name","Field type","Default value","Variable to save field value",Column)
Available in: Professional edition

This command adds a field to a form.

#	Parameter name	Parameter description
1	Form identifier	An identifier of the form (for example, "FM1") to which the item will be added.
2	Field name	The label that appears above the control in the form window.
3	Field type	Item type can be one of these: "LIST" - selection from list of values defined as a string of values delimited by | character. Example: Item1|Item2|Item3 "EDIT" - edit control (single line) "EDIT_ML5" - edit control (5 lines) "EDIT_ML10" - edit control (10 lines) "CHECK" - check box button "LINE" - static line "TEXT" - static text "BUTTON" - button "PWD" - password input box "RADIO" - radio button "EDIT_FILE" - file path "EDIT_FOLDER" - folder path
4	Default value	Defines default the value (the value that initially appears in the form). Check button (CHECK) and radio button (RADIO) have YES or NO values.
5	Variable to save field value	Variable that receives the data (value) the user enters. Check button (CHECK) and radio button (RADIO) have YES or NO values. If the item type is "LIST" then there is automatically created also variable that contains index of the item selected. For example, if the variable to receive selected "LIST" value is "vListValue" then - after the form is closed by user by clicking OK button - there is also variable "vListValue_Inx" that contains index of the item selected. The index numbers start from 0.
6	Column	

Example (Macro Steps):

```
1  <#> <#> This example shows how to use form commands
2  Macro execution: ONLY COMMANDS
3  Form FIELD "This is a simple calculator for: + - * /" of type "Static text" (Default value=0, Variable to save field value=, Form identifier=FM1)
4  Form FIELD "" of type "Separator" (Default value=0, Variable to save field value=, Form identifier=FM1)
5  Form FIELD "Operand 1:" of type "Text edit" (Default value=0, Variable to save field value=vOper1, Form identifier=FM1)
6  Form FIELD "Operation:" of type "Drop down list" (Default value=Plus|Minus|Multiply|Divide, Variable to save field value=vOperation, Form identifier=FM1)
7  Form FIELD "Operand 2:" of type "Text edit" (Default value=0, Variable to save field value=vOper2, Form identifier=FM1)
8  Form FIELD "" of type "Separator" (Default value=0, Variable to save field value=, Form identifier=FM1)
9  Form FIELD "Continue ?" of type "Check box" (Default value=YES, Variable to save field value=vAgain, Form identifier=FM1)
10 Jump TARGET "lbl_Again"
11 Form OPEN "FM1", Window title="Simple Calculator"
12 IF STRING _vCanceled==1
13     Macro EXIT
14 ENDIF
15 IF STRING vOperation==Plus
16     Variable SET "vOper1=EXPR(%vOper1%+%vOper2%)", Message text=""
17 ENDIF
18 IF STRING vOperation==Minus
19     Variable SET "vOper1=EXPR(%vOper1%-%vOper2%)", Message text=""
20 ENDIF
21 IF STRING vOperation==Multiply
22     Variable SET "vOper1=EXPR(%vOper1%*%vOper2%)", Message text=""
23 ENDIF
24 IF STRING vOperation==Divide
25     Variable SET "vOper1=EXPR03(%vOper1%/%vOper2%)", Message text=""
26 ENDIF
27 Message SHOW "Information" : "vOper1" (other parameters: x = -100, y = -100, Window title = The result is:,
```

Example (Plain Text):

```
<#> This example shows how to use form commands
<cmds>
 <form_item>("FM1","This is a simple calculator for: + - * /","TEXT","0","",1)
<form_item>("FM1","","LINE","0","",1)
<form_item>("FM1","Operand 1:","EDIT","0","vOper1",1)
<form_item>("FM1","Operation:","LIST","Plus|Minus|Multiply|Divide","vOperation",1)
<form_item>("FM1","Operand 2:","EDIT","0","vOper2",1)
<form_item>("FM1","","LINE","0","",1)
<form_item>("FM1","Continue ?","CHECK","YES","vAgain",1)
<label>("lbl_Again")

<form_show>("FM1","Simple Calculator","calc.exe",0,500,0,,,1,1)
<if_str>("_vCanceled==1") <exitmacro> <endif>

<if_str>("vOperation==Plus")<#>
<varset>("vOper1=EXPR(%vOper1%+%vOper2%)","")
<endif>
<if_str>("vOperation==Minus")
   <varset>("vOper1=EXPR(%vOper1%-%vOper2%)","")
<endif>
<if_str>("vOperation==Multiply")
   <varset>("vOper1=EXPR(%vOper1%*%vOper2%)","")
<endif>
<if_str>("vOperation==Divide")
   <varset>("vOper1=EXPR03(%vOper1%/%vOper2%)","")
<endif>
<msg>(-100,-100,"vOper1","The result is:",1,,0)
<if_str>("vAgain==YES")
   <goto>("lbl_Again")
<endif>
```

Menu ADD ITEM - < menu_additem >() ... [Pro]

Menu ADD ITEM
<menu_additem>("Item name","Item identifier","Icon file",Icon index)
Available in: Professional edition

This command adds a new item to menu shown on the screen using "menu_show" command.

#	Parameter name	Parameter description
1	Item name	Name of the item as it shows in the menu. To add submenu items insert '.' (dot) to begin of the item name. To specify end of the submenu, insert just '.' Item. To underline a specific character within the item, insert '&' character right before it.
2	Item identifier	This is an optional parameter. If it is supplied, the "menu_show" command returns item identifier rather than item name.
3	Icon file	File containing an icon (.exe, .dll, .ico).
4	Icon index	Position of the icon in the file.

Example (Macro Steps):

```
1   <#> <#> This macro will show how to use 'menu_additem'
2   <#> <#> and 'menu_show' commands
3   Macro execution: ONLY COMMANDS
4   Menu ADD ITEM "Item 1", Item identifier=""
5   Menu ADD ITEM "Item 2", Item identifier=""
6   Menu ADD ITEM "Item 3", Item identifier=""
7   Menu ADD ITEM ".Item 1 - 1", Item identifier=""
8   Menu ADD ITEM ".Item 1 - 2", Item identifier=""
9   Menu ADD ITEM ".Item 1 - 3", Item identifier=""
10  Menu ADD ITEM ".Item 1 - 4", Item identifier=""
11  Menu ADD ITEM ".", Item identifier=""
12  Menu SHOW [ x = -1, y = -1 ], Variable to keep selected item = vItem, Get item = Identifier, Add item numbers (1-9, A-Z) = Yes
13  IF STRING vItem != NO
14      Message SHOW "" : "Selected item is: %vItem%" (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
15  ENDIF
```

Example (Plain Text):

```
<#> This macro will show how to use 'menu_additem'
<#> and 'menu_show' commands
<cmds>
<menu_additem>("Item 1","","shell32.dll",3)
<menu_additem>("Item 2","","shell32.dll",4)
<menu_additem>("Item 3","","shell32.dll",5)
<menu_additem>(".Item 1 - 1","","shell32.dll",6)
<menu_additem>(".Item 1 - 2","","shell32.dll",7)
<menu_additem>(".Item 1 - 3","","shell32.dll",8)
<menu_additem>(".Item 1 - 4","","shell32.dll",9)
<menu_additem>(".")
<menu_show>(-1,-1,vItem,1,1)
<if_str>("vItem != NO")
   <msg>(-100,-100,"Selected item is: %vItem%","Message",1)
<endif>
```

Menu SHOW - < menu_show >() ... [Pro]

Menu SHOW
<menu_show>(x,y,Variable to keep selected item,Get item,Add item numbers (1-9, A-Z))
Available in: Professional edition

Opens menu that contains items previously added by "menu_additem" command.

#	Parameter name	Parameter description
1	x	X-coordinate of the menu position. If Xpos is 0 and Ypos is also 0, the menu appears on caret position where text is entered. If Xpos is â€“1 and Ypos is also â€“1, the menu appears on mouse cursor position.
2	y	Y-coordinate of the menu position. If Xpos is 0 and Ypos is also 0, the menu appears on caret position where text is entered. If Xpos is â€“1 and Ypos is also â€–1, the menu appears on mouse cursor position.
3	Variable to keep selected item	This variable receives the menu item user clicks on. If no menu item is selected, the variable receives "NO" value.
4	Get item	If this parameter is 1, the Variable receives the item name as it was added using "menu_additem" command. If this parameter is 0, the Variable receives the order number of the item clicked.
5	Add item numbers (1-9, A-Z)	Can be one of these values: 0 - default behavior. 1 - a prefix (1-9, a-z) is added before each macro name in the menu. This allows user to run macro by pressing the prefix key.

Example (Macro Steps):

```
1   <#> <#> This macro will show how to use 'menu_additem'
2   <#> <#> and 'menu_show' commands
3   Macro execution: ONLY COMMANDS
4   Menu ADD ITEM "White", Item identifier=""
5   Menu ADD ITEM "Gray", Item identifier=""
6   Menu ADD ITEM "Black", Item identifier=""
7   Menu ADD ITEM ".RGB", Item identifier=""
8   Menu ADD ITEM ".Red", Item identifier=""
9   Menu ADD ITEM ".Green", Item identifier=""
10  Menu ADD ITEM ".Blue", Item identifier=""
11  Menu ADD ITEM ".", Item identifier=""
12  Menu SHOW [ x = -1, y = -1 ], Variable to keep selected item = vColor, Get item = Identifier, Add item
    numbers (1-9, A-Z) = Yes
13  IF STRING vColor != NO
14      Message SHOW "" : "Selected item is: %vColor%" (other parameters: x = -100, y = -100, Window title =
        Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
15  ENDIF
```

Example (Plain Text):

```
<#> This macro will show how to  use 'menu_additem'
<#> and 'menu_show' commands
<cmds>
<menu_additem>("White")
<menu_additem>("Gray")
<menu_additem>("Black")
<menu_additem>(".RGB")
<menu_additem>(".Red")
<menu_additem>(".Green")
<menu_additem>(".Blue")
<menu_additem>(".")

<menu_show>(-1,-1,vColor,1,1)

<if_str>("vColor != NO")
   <msg>(-100,-100,"Selected item is: %vColor%","Message",1)
<endif>
```

Menu of MACROS - < macromenu >() ... [Pro]

Menu of MACROS
<macromenu>(x,y,"Macro group",Option)
Available in: Professional edition

This command shows a menu consisting from (enabled) macros found in the defined macro group. Clicking on a menu item (macro) will cause that the macro is started.

#	Parameter name	Parameter description
1	x	X-coordinate of the menu position. If Xpos is 0 and Ypos is also 0, the menu appears on caret position where text is entered. If Xpos is -1 and Ypos is also -1, the menu appears on mouse cursor position. If Xpos is -2 and Ypos is also -2, the menu appears in the center of the currently active window or in the center of screen.
2	y	Y-coordinate of the menu position. If Xpos is 0 and Ypos is also 0, the menu appears on caret position where text is entered. If Xpos is -1 and Ypos is also -1, the menu appears on mouse cursor position. If Xpos is -2 and Ypos is also -2, the menu appears in the center of the currently active window or in the center of screen.
3	Macro group	Macros (name of the macro) from this group will show in the menu.
4	Option	Can be one of these values: 2 - default behavior. 3 - a prefix (1-9, a-z) is added before each macro name in the menu. This allows user to run macro by pressing the prefix key. The macro menu is by default sorted alphabetically. If the +4 are added to the options above (so that the options number will be either 6 or 7) then the macro menu is to sorted based on the "Order" column in the macro listing in main window.

Example (Macro Steps):

1 **<#> <#> This macro will open menu with macros from**

2 **<#> <#> 'New Macro Group' group**

3 **Macro execution: ONLY COMMANDS**

4 **Menu of MACROS** from group "New Macro Group", [x = -1, y = -1], Option = Sort alphabetically+Add item numbering

Example (Plain Text):

```
<#> This macro will open menu with macros from
<#> 'New Macro Group' group
<cmds>
<macromenu>(-1,-1,"New Macro Group",3)
```

Window Manipulation

ACTIVATE - < actwin >() ... [Free]

Window ACTIVATE
<actwin>("Window",Match,Window state)
Available in: Free edition

Activates specified window. If the window doesn't exist a macro can be started (to open desired application or notify user, for example).

#	Parameter name	Parameter description
1	Window	Window identifier in form of Window Identification Path (WIP) or HWND. It identifies the window that is to be activated.
2	Match	Takes effect only if a window is identified using WIP parameter. Can be one of these values: 0 - match substrings in WIP 1 - match exact strings in WIP
3	Window state	State of the window after the window is activated: 0 - Normal 1 - Minimized 2 - Maximized 3 - Let the state unchanged

Example (Macro Steps):

1 <#> <#> This macro activates "Notepad" window if it is opened

2 **Macro execution: ONLY COMMANDS**

3 **IF WINDOW** "Notepad" Is Open (Match=Partial)

4 **Window ACTIVATE** bring "Notepad" window to top (other parameters: Match = Partial, Window state = Normal, %p4_name = no)

5 **ELSE** activate

6 **Message SHOW** "" : "'Notepad' is not opened!" (other parameters: x = 100, y = 100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

7 **ENDIF**

Example (Plain Text):

```
<#> This macro activates "Notepad" window if it is opened
<#>
<cmds>

<if_win>("Notepad","OPEN",0)
  <actwin>("Notepad",0,0,"no")
<else>
  <msg>(100,100,"'Notepad' is not opened!","Message",1)
<endif>
```

MOVE - < winmove >() ... [Pro]

Window MOVE
<winmove>("Window",Match,x,y)
Available in: Professional edition

The command moves specified window to defined position.

#	Parameter name	Parameter description
1	Window	Window identifier in form of Window Identification Path (WIP) or HWND. It identifies the window that is to be moved.
2	Match	Takes effect only if a window is identified using WIP parameter. Can be one of these values: 0 - match substrings in WIP 1 - match exact strings in WIP
3	x	X-coordinate of desired window position
4	y	Y-coordinate of desired window position

Example (Macro Steps):

```
1    <#> <#> This macro moves "Notepad" window to position (32,32)
2    Macro execution: ONLY COMMANDS
3    IF WINDOW "[*Notepad|Notepad|#0|#0]" Is Open (Match=Partial)
4        Window MOVE "[*Notepad|Notepad|#0|#0]" to [ x=32, y=32 ]
5    ELSE activate
6        Message SHOW "" : "'Notepad' is not opened!" (other parameters: x = 100, y = 100, Window title =
         Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
7    ENDIF
```

Example (Plain Text):

```
<#> This macro moves "Notepad" window to position (32,32)
<#>
<cmds>
<if_win>("[*Notepad|Notepad|#0|#0]","OPEN",0)
  <winmove>("[*Notepad|Notepad|#0|#0]",0,32,32)
<else>
  <msg>(100,100,"'Notepad' is not opened!","Message",1)
<endif>
```

RESIZE - < winresize >() ... [Pro]

Window RESIZE
<winresize>("Window",Match,Width,Height)
Available in: Professional edition

The command resizes specified window.

#	Parameter name	Parameter description
1	Window	Window identifier in form of Window Identification Path (WIP) or HWND. It identifies the window that is to be closed.
2	Match	Takes effect only if a window is identified using WIP parameter. Can be one of these values: 0 - match substrings in WIP 1 - match exact strings in WIP
3	Width	Desired window's width
4	Height	Desired window's height

Example (Macro Steps):

```
1   <#> <#> This macro resizes "Notepad" window to size 300x200
2   Macro execution: ONLY COMMANDS
3   IF WINDOW "[*Notepad|Notepad|#0|#0]" Is Open (Match=Partial)
4       Window RESIZE "[*Notepad|Notepad|#0|#0]" to [ Width=300, Height=200 ]
5   ELSE activate
6       Message SHOW "" : "'Notepad' is not opened!" (other parameters: x = 100, y = 100, Window title =
        Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
7   ENDIF
```

Example (Plain Text):

```
<#> This macro resizes "Notepad" window to size 300x200
<#>
<cmds>

<if_win>("[*Notepad|Notepad|#0|#0]","OPEN",0)
  <winresize>("[*Notepad|Notepad|#0|#0]",0,300,200)
<else>
  <msg>(100,100,"'Notepad' is not opened!","Message",1)
<endif>
```

Minimize All - < winminall > ... [Pro]

Window Minimize All
<winminall>
Available in: Professional edition

The command minimizes all windows.

Example (Macro Steps):

1 **<#> <#> This macro minimizes all windows**

2 **Window Minimize All**

Example (Plain Text):

```
<#> This macro minimizes all windows
<#>
<winminall>
```

CLOSE - < winclose >() ... [Pro]

Window CLOSE
<winclose>("Window",Match)
Available in: Professional edition

This command closes specified window. If the window is main application window this command causes the application is closed.

#	Parameter name	Parameter description
1	Window	Window identifier in form of Window Identification Path (WIP) or HWND. It identifies the window that is to be closed.
2	Match	Takes effect only if a window is identified using WIP parameter. Can be one of these values: 0 - match substrings in WIP 1 - match exact strings in WIP

Example (Macro Steps):

```
1   <#> <#> This macro closes "Notepad" window
2   Macro execution: ONLY COMMANDS
3   IF WINDOW "[* - Notepad|Notepad|#566|#124]" Is Open (Match=Partial)
4       Window CLOSE "[* - Notepad|Notepad|#566|#124]"
5   ELSE activate
6       Message SHOW "" : "'Notepad' is not opened!" (other parameters: x = 100, y = 100, Window title =
        Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
7   ENDIF
```

Example (Plain Text):

```
<#> This macro closes "Notepad" window
<#>
<cmds>

<if_win>("[* - Notepad|Notepad|#566|#124]","OPEN",0)
  <winclose>("[* - Notepad|Notepad|#566|#124]",0)
<else>
  <msg>(100,100,"'Notepad' is not opened!","Message",1)
<endif>
```

CHANGE STATE - < winstate >() ... [Pro]

Window CHANGE STATE
<winstate>("Window","Window state")
Available in: Professional edition

This command changes specified window state.

#	Parameter name	Parameter description
1	Window	Window identifier in form of Window Identification Path (WIP) or HWND. It identifies the window that state is to be changed.
2	Window state	Can be one of these values: "MIN" "RESTORE" "MAX" "ALWAYS_TOP" "NOT_ALWAYS_TOP" "HIDE" "SHOW"

Example (Macro Steps):

```
1   <#> <#> This macro minimizes "Notepad" window
2   Macro execution: ONLY COMMANDS
3   IF WINDOW "[* - Notepad|Notepad|#0|#119]" Is Open (Match=Partial)
4       Window CHANGE STATE Window=[* - Notepad|Notepad|#0|#119], Window state=Minimize
5   ELSE activate
6       Message SHOW "" : "'Notepad' is not opened!" (other parameters: x = 100, y = 100, Window title =
        Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
7   ENDIF
```

Example (Plain Text):

```
<#> This macro minimizes "Notepad" window
<cmds>
<if_win>("[* - Notepad|Notepad|#0|#119]","OPEN",0)
  <winstate>("[* - Notepad|Notepad|#0|#119]","MIN")
<else>
  <msg>(100,100,"'Notepad' is not opened!","Message",1)
<endif>
```

ENUMERATE - < win_enumerate >() ... [Pro]

Window ENUMERATE
<win_enumerate>(Variable array for enumerated items,Variable array size,Match)
Available in: Professional edition

This command enumerates windows.

#	Parameter name	Parameter description
1	Variable array for enumerated items	Variable (array) that receives window title of each visible window.
2	Variable array size	Number of visible windows saved in VarWindows variable.
3	Match	0 - enumerate only visible windows and insert window titles to VarWindows variable 1 - enumerate both visible and hidden windows and insert window titles to VarWindows variable 2 - enumerate only visible windows and insert HWND to VarWindows variable 3 - enumerate both visible and hidden windows and insert HWND to VarWindows variable

Example (Macro Steps):

1 <#> <#> This macro enumerates opened windows

2 **Macro execution: ONLY COMMANDS**

3 **Window ENUMERATE** "Only visible windows, get window titles", Variable array for enumerated items = vWin, Variable array size = vNum

4 **Loop BEGIN** Repeat = vNum

5 **Message SHOW** "" : "%vWin[_vLoopCounter0]%" (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout (seconds) = , Always on top =).

6 **Loop END**

Example (Plain Text):

```
<#> This macro enumerates opened windows
<#>
<cmds>

<win_enumerate>(vWin,vNum,0)
<begloop>(vNum)
  <msg>(-100,-100,"%vWin[_vLoopCounter0]%","Message",1)
<endloop>
```

INFO - < wininfo >() ... [Pro]

Window INFO
<wininfo>("Window","Unused","Match")
Available in: Professional edition

This command retrieves information about required window. After the command is processed, required information is saved in following system variables:

_vWinRectX1 - X position of upper left corner of the window in screen coordinates
_vWinRectY1 - Y position of upper left corner of the window in screen coordinates
_vWinRectX2 - X position of lower right corner of the window in screen coordinates
_vWinRectY2 - Y position of lower right corner of the window in screen coordinates
_vWinWdt - window width
_vWinHgt - window height
_vWinTitle - window title
_vWinClass - window class
_vWinState - window state (MIN, RESTORE, MAX, HIDDEN)
_vWinActive - YES, if window is active (on top receiving keyboard input), otherwise NO
_vWinHWND - HWND of the window

#	Parameter name	Parameter description
1	Window	Window identifier in form of Window Identification Path or HWND. It identifies window the information is to be retrieved from.
2	Unused	Must be empty.
3	Match	Takes effect only if a window is identified using WIP parameter. Can be one of these values: 0 - match substrings in WIP 1 - match exact strings in WIP

Example (Macro Steps):

```
1   <#> <#> This macro retrieves information about "Notepad" window
2   Macro execution: ONLY COMMANDS
3   IF WINDOW "Notepad" Is Open (Match=Partial)
4       Window INFO retrieve information about window "Notepad" (Match = Partial)
5       Message SHOW "" : "Window information: %_vWinTitle% %_vWinClass% %_vWinActive%
        %_vWinState%" (other parameters: x = -100, y = -100, Window title = Message, Buttons = OK, Timeout
        (seconds) = , Always on top = ).
6   ELSE activate
7       Message SHOW "" : "'Notepad' is not opened!" (other parameters: x = 100, y = 100, Window title =
        Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
8   ENDIF
```

Example (Plain Text):

```
<#> This macro retrieves information about "Notepad" window
```

```
<#>
<cmds>

<if_win>("Notepad","OPEN",0)
   <wininfo>("Notepad","",0)
   <msg>(-100,-100,"Window information:

%_vWinTitle%

%_vWinClass%

%_vWinActive%

%_vWinState%","Message",1)

<else>
   <msg>(100,100,"'Notepad' is not opened!","Message",1)
<endif>
```

Image FIND in WINDOW - < win_findimage >() ... [Pro]

Window Image FIND in WINDOW

<win_findimage>("Window",Match,"Image file",Start search X,Start search Y,Variable for image found X,Variable for image found Y,Image match,Search area width,Search area height)
Available in: Professional edition

This command searches for defined image(s) within the given window. If the image is found in the window the command sets supplied coordinate variables. If multiple image files are defined using wildcards then [x,y] position variables for each image are created for each image file. For example, if image files are defined using "c:\find_images*.bmp" and there are "Button.bmp" and "Title.bmp" image files in the "c:\find_images" folder then variables "button.bmp_x" and "button.bmp_y" variables that defines position of the "Button.bmp" image on the screen is created. The same for "title.bmp" file there are "title.bmp_x" and "title.bmp_y" variable created. Using such variables it is possible to determine what images were found and what are their position on the screen. Image file names are always converted to lowercase.

Important:

It is application specific how the window content is drawn within its window and in some cases what is visible on the computer screen as a one image is actually composed by multiple pieces. For this reason, use <display_findimage> command if this command is not working with a particular application.

The bitmap file the command is finding must be captured with the same DPI (or zoom in web browser) as the content presented on the screen. These are typical problems why the command fails:
1. The bitmap file is captured on a monitor with higher/lower DPI than the monitor where the command is finding the image. To prevent this problem always capture the image on the same monitor where the command is executed.

2. The bitmap file is captured in web browser with different zoom setting than the current zoom. To prevent this use the same web browser and the same zoom setings when capturing image and running the macro.

#	Parameter name	Parameter description
1	Window	Window Identification Path or HWND of the window where the image is to be searched in. HWND is a unique handle Windows internally uses to identify each window. The HWND can be retrieved by some commands (,) or is provided by some system variables (_vKeybdFocusWindow_HWND, _vActiveWindow_HWND, _vActiveWindowPrev_HWND).
2	Match	Takes effect only if a window title is used as WinTitleOrHWND parameter. Can be one of these values: 0 - WinTitle can be substring of a window title 1 - WinTitle must exactly match a window title
3	Image file	(Full) path to the image file. This is a bitmap image that is captured using "Capture..." feature in the "win_findimage" command editor. Multiple image files are supported using wildcards (* or ?).
4	Start search X	X-coordinate where to start searching in the window.
5	Start search Y	Y-coordinate where to start searching in the window.
6	Variable for image found X	It is possible to scope searching to an area smaller than the whole window. This attribute specifies the width of the searching area. If this parameter is set to "0" then the whole window width is being searched.
7	Variable for image found Y	It is possible to scope searching to an area smaller than the whole window. This attribute specifies the height of the searching area. If this parameter is set to "0" then the whole window height is being searched.
8	Image match	Name of the variable that receives X-coordinate of the position of the image in the window. If the image is not found then the variable receives "-1".
9	Search area width	Name of the variable that receives Y-coordinate of the position of the image in the window. If the image is not found then the variable receives "-1".
10	Search area height	If 0 then the image does not has to exactly match, a certain level of tolerance is allowed. If 1 then the image has to exactly match.

Example (Macro Steps):

```
1   <#> <#> This macro finds a "sin" button in Calculator window and clicks on it

2   Key Enter

3   Macro execution: ONLY COMMANDS

4   Window Image FIND in WINDOW "[Calculator|CalcFrame|#25|#543]" (Match = Partial, Image file =
    C:\Temp\sin.bmp, Start search X = 0, Start search Y = 0, Variable for image found X = vSinBtnX, Variable for
    image found Y = vSinBtnY, Image match = Tolerant, Search area width = 0, Search area height = 0)

5   IF vSinBtnX > -1

6       Window ACTIVATE bring "[Calculator|CalcFrame|#0|#0]" window to top (other parameters: Match =
        Partial, Window state = Normal, %p4_name = no)

7       Mouse COORDINATES are now RELATIVE to active WINDOW

8       Mouse MOVE position [ x=vSinBtnX, y=vSinBtnY ]

9       Mouse BUTTON: LEFT button DOWN

10      Mouse BUTTON: LEFT button UP

11  ENDIF
```

Example (Plain Text):

```
<#> This macro finds a "sin" button in Calculator window and clicks on it
<#>

<cmds>

<win_findimage>("[Calculator|CalcFrame|#25|#543]",0,"C:\Temp\sin.bmp",0,0,vSinBtnX,vSinBtnY,0,0,0)

<if>("vSinBtnX > -1")

  <actwin>("[Calculator|CalcFrame|#0|#0]",0,0,"no")
  <mousemove_relative_win>
  <mm>(vSinBtnX,vSinBtnY)<mlbd><mlbu>

<endif>
```

Image CAPTURE from WINDOW - < win_captureimage >() ... [Pro]

Window Image CAPTURE from WINDOW
<win_captureimage>("Window",Match,x,y,Width,Height,"Image file")
Available in: Professional edition

This command captures an image in defined window.

Important:

It is application specific how the window content is drawn within its window and in some cases what is visible on the computer screen as a one image is actually composed by multiple pieces. For this reason, use <display_captureimage> command if this command is not working with a particular application.

#	Parameter name	Parameter description
1	Window	Window identifier in form of Window Identification Path or HWND. It identifies the window where an image is to be captured.
2	Match	Takes effect only if a window is identified using WIP parameter. Can be one of these values: 0 - match substrings in WIP 1 - match exact strings in WIP
3	x	X coordination of the upper left corner of the area to be captured.
4	y	Y coordination of the upper left corner of the area to be captured.
5	Width	Width of the area to be captured. If it is set to "0" then whole window is captured.
6	Height	Height of the area to be captured.
7	Image file	Name (or full path) of the resulting image file.

Example (Macro Steps):

1 <#> <#> This macro shows how to capture picture of an active window

2 **Macro execution: ONLY COMMANDS**

3 **Window Image CAPTURE from WINDOW** Window = "_vActiveWindow_HWND", Match = "Partial", x = "0", y = "0", Width = "0", Height = "0", Image file = "%_vFolder_Personal%\WindowImage.bmp"

4 **Message SHOW** "Question" : "Do you want to see the captured image now?" (other parameters: x = -100, y = -100, Window title = Message, Buttons = Yes and No, Timeout (seconds) = 0, Always on top =).

5 **IF** _vMsgButton==YES

6 **File OPEN** open file "%_vFolder_Personal%\WindowImage.bmp" in system default viewer.

7 **ENDIF**

Example (Plain Text):

<#> This macro shows how to capture picture of an active window
<cmds>

<win_captureimage>("_vActiveWindow_HWND",0,0,0,0,0,"%_vFolder_Personal%\WindowImage.bmp")

```
<msg>(-100,-100,"Do you want to see the captured image now?","Message",2,0,1)

<if>("_vMsgButton==YES")
   <fileopen>("%_vFolder_Personal%\WindowImage.bmp",0)
<endif>
```

APPLICATION - < actapp >() ... [Free]

Window APPLICATION
<actapp>("Window")
Available in: Free edition

Activates application the specified window belongs to. The command does not change what window in the application is active - the last active application's window remains active. This is the difference to what command does.

#	Parameter name	Parameter description
1	Window	Window identifier in form of Window Identification Path (WIP) or HWND. It identifies window of the application to be activated.

Example (Macro Steps):

```
1    <#> <#> This macro activates "Notepad" application if it is opened
2    Macro execution: ONLY COMMANDS
3    IF WINDOW "[*|Notepad|#0|#0]" Is Open (Match=Partial)
4        Window APPLICATION activate [*|Notepad|#0|#0]
5    ELSE activate
6        Message SHOW "" : "'Notepad' is not opened!" (other parameters: x = 100, y = 100, Window title =
         Message, Buttons = OK, Timeout (seconds) = , Always on top = ).
7    ENDIF
```

Example (Plain Text):

```
<#> This macro activates "Notepad" application if it is opened
<#>
<cmds>

<if_win>("[*|Notepad|#0|#0]","OPEN",0)
  <actapp>("[*|Notepad|#0|#0]")
<else>
  <msg>(100,100,"'Notepad' is not opened!","Message",1)
<endif>
```

Xml Parser

File Open - < xml_file_open >() ... [Pro]

XML File Open
<xml_file_open>("File path",File handle variable,Root element handle variable,"Password")
Available in: Professional edition

This command opens an Xml file and parses it. If the file content is not valid Xml then the open command fails. The command returns file handle and root element handle (to variables passed as parameters) needed for other operarions.

#	Parameter name	Parameter description
1	File path	Xml file path.
2	File handle variable	Xml file handle - a unique number that is used as parameter to other commands (for eaxample, to close the Xml file).
3	Root element handle variable	Xml root element handle - a unique number that identifies the root element of the Xml file. It is used as input to other commands (for example, to get child elements and traverse the Xml file.
4	Password	Password used to decrypt the data. The password must be the same that was previously used to encrypt data (in 'xml_file_create'command). If incorrect password is provided then commands that reads data from the XML file ('xml_element_get', 'xml_attribute_get' or 'xml_findtext') fails. Leave the parameter empty if the data are not encrypted.

Example (Macro Steps):

1 <#> <#>This macro shows how to open and close an Xml file

2 **Macro execution: ONLY COMMANDS**

3 XML File Open File path = "c:\myXmlFileLocation\file.xml", File handle variable = "vXmlDoc", Root element handle variable = "vXmlRoot", Password = ""

4 <#> <#> Do something useful with the Xml file...

5 XML File Close "vXmlDoc"

Example (Plain Text):

```
<#>This macro shows how to open and close an Xml file
<cmds>
<xml_file_open>("c:\myXmlFileLocation\file.xml",vXmlDoc,vXmlRoot)
<#> Do something useful with the Xml file...
<xml_file_close>(vXmlDoc)
```

File Save - < xml_file_save >() ... [Pro]

XML File Save
<xml_file_save>(File handle variable,"File path")
Available in: Professional edition

This command saves an Xml file to disk.

#	Parameter name	Parameter description
1	File handle variable	Xml file handle - an Xml file identifier provided from "open xml" or "create xml" commands.
2	File path	Name (full path) of the file.

Example (Macro Steps):

```
1    <#> <#>This macro shows how to save a created Xml file
2    Macro execution: ONLY COMMANDS
3    XML File Create vXmlDoc, Root element handle variable = vXmlRoot,Password =
4    XML File Save "vXmlDoc" (File path = "c:\temp\newFile.xml")
5    XML File Close "vXmlDoc"
```

Example (Plain Text):

```
<#>This macro shows how to save a created Xml file
<cmds>
<xml_file_create>(vXmlDoc,vXmlRoot)
<xml_file_save>(vXmlDoc,"c:\temp\newFile.xml")
<xml_file_close>(vXmlDoc)
```

Element Get - < xml_element_get >() ... [Pro]

XML Element Get

<xml_element_get>(Element handle variable,What,Variable receiving result)
Available in: Professional edition

This command provides access to an Xml file element information such as name, text, child elements, etc.

#	Parameter name	Parameter description
1	Element handle variable	Element handle - variable that identifies the element to get information from.
2	What	Type of information to retrieve: SIBLING_NEXT - identifier of next sibling element. SIBLING_PREV - identifier of previous sibling element. CHILD_FIRST - identifier of first child element. CHILD_LAST - identifier of last child element. TEXT - the element text. NAME - the element name. PATH - the element path in the XML from root. For example, "contacts\contact\address\street" for the "street" element that is child of the "address" element...
3	Variable receiving result	Name of variable that will receive retrieved information.

Example (Macro Steps):

```
<#> <#>This macro shows how to work with Xml file content
<#> <#>Create XML file with some simple content and save it to disk
Macro execution: ONLY COMMANDS
XML File Create vXmlDoc, Root element handle variable = vXmlRoot,Password =
XML Element Create (File handle variable = "vXmlDoc", Name = "elem1", Text = "This is elem1 text",
Element handle variable = "vElem1")
XML Attribute Set (Element handle variable = "vElem1", Attribute name = "a1", Value = "Hello")
XML Element Set "vXmlRoot" (What = CHILD_FIRST, Input = vElem1)
XML File Save "vXmlDoc" (File path = "%TEMP%\test.xml")
XML File Close "vXmlDoc"
<#> <#>Open the XML file previously created and explore the contet
XML File Open File path = "%TEMP%\test.xml", File handle variable = "v2XmlDoc", Root element handle
variable = "v2XmlRoot", Password = ""
IF STRING %v2XmlDoc%==%_vStrEmpty%
    Message SHOW "Information" : "Xml file open failed." (other parameters: x = -100, y = -100, Window title =
    Error, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
ENDIF
XML Element Get "v2XmlRoot" (What = CHILD_FIRST, Variable receiving result = v2ElemChild1)
IF STRING %v2ElemChild1%==%_vStrEmpty%
    Message SHOW "Information" : "Xml element not found." (other parameters: x = -100, y = -100, Window
    title = Error, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
ENDIF
XML Element Get "v2ElemChild1" (What = NAME, Variable receiving result = v2ElemName)
XML Element Get "v2ElemChild1" (What = TEXT, Variable receiving result = v2ElemText)
XML Attribute Get (Element handle variable = "%v2ElemChild1%", Attribute name = "a1", Variable for result =
"v2A1AttributeValue")
XML Navigate to Element (Element handle variable = "v2XmlDoc", Path = "root\elem1", Element handle
variable = "v2ElemNavigated")
IF STRING %v2ElemNavigated%==%_vStrEmpty%
    Message SHOW "Information" : "Xml file open failed." (other parameters: x = -100, y = -100, Window title =
    Error, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
ENDIF
XML File Close "v2XmlDoc"
```

Example (Plain Text):

```
<#>This macro shows how to work with Xml file content
<#>Create XML file with some simple content and save it to disk
<cmds>
<xml_file_create>(vXmlDoc,vXmlRoot)
<xml_element_create>(vXmlDoc,elem1,This is elem1 text,vElem1)
<xml_attribute_set>(vElem1,a1,Hello)
<xml_element_set>(vXmlRoot,CHILD_FIRST,vElem1)
<xml_file_save>(vXmlDoc,"%TEMP%\test.xml")
<xml_file_close>(vXmlDoc)<#>Open the XML file previously created and explore the contet
<xml_file_open>("%TEMP%\test.xml",v2XmlDoc,v2XmlRoot)
<if_str>("%v2XmlDoc%==%_vStrEmpty%")
<msg>(-100,-100,"Xml file open failed.","Error",1,0,0,0)
<endif>
<xml_element_get>(v2XmlRoot,CHILD_FIRST,v2ElemChild1)
<if_str>("%v2ElemChild1%==%_vStrEmpty%")
<msg>(-100,-100,"Xml element not found.","Error",1,0,0,0)
<endif>
<xml_element_get>(v2ElemChild1,NAME,v2ElemName)
<xml_element_get>(v2ElemChild1,TEXT,v2ElemText)
<xml_attribute_get>(%v2ElemChild1%,a1,v2A1AttributeValue)
<xml_element_navigate>(v2XmlDoc,"root\elem1",v2ElemNavigated)
<if_str>("%v2ElemNavigated%==%_vStrEmpty%")
<msg>(-100,-100,"Xml file open failed.","Error",1,0,0,0)
<endif>
<xml_file_close>(v2XmlDoc)
```

Attribute Get - < xml_attribute_get >() ... [Pro]

XML Attribute Get
<xml_attribute_get>(Element handle variable,Attribute name,Variable for result)
Available in: Professional edition

This command retrieves given Xml file element attribute value.

#	Parameter name	Parameter description
1	Element handle variable	Element handle - variable that identifies the element.
2	Attribute name	Attribute name.
3	Variable for result	Name of variable that will receive attribute value. If the attribute is not found then the value is an empty string.

Example (Macro Steps):

```
<#> <#>This macro shows how to work with Xml file content
<#> <#>Create XML file with some simple content and save it to disk
Macro execution: ONLY COMMANDS
XML File Create vXmlDoc, Root element handle variable = vXmlRoot,Password =
XML Element Create (File handle variable = "vXmlDoc", Name = "elem1", Text = "This is elem1 text",
Element handle variable = "vElem1")
XML Attribute Set (Element handle variable = "vElem1", Attribute name = "a1", Value = "Hello")
XML Element Set "vXmlRoot" (What = CHILD_FIRST, Input = vElem1)
XML File Save "vXmlDoc" (File path = "%TEMP%\test.xml")
XML File Close "vXmlDoc"
<#> <#>Open the XML file previously created and explore the contet
XML File Open File path = "%TEMP%\test.xml", File handle variable = "v2XmlDoc", Root element handle
variable = "v2XmlRoot", Password = ""
IF STRING %v2XmlDoc%==%_vStrEmpty%
    Message SHOW "Information" : "Xml file open failed." (other parameters: x = -100, y = -100, Window title =
    Error, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
ENDIF
XML Element Get "v2XmlRoot" (What = CHILD_FIRST, Variable receiving result = v2ElemChild1)
IF STRING %v2ElemChild1%==%_vStrEmpty%
    Message SHOW "Information" : "Xml element not found." (other parameters: x = -100, y = -100, Window
    title = Error, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
ENDIF
XML Element Get "v2ElemChild1" (What = NAME, Variable receiving result = v2ElemName)
XML Element Get "v2ElemChild1" (What = TEXT, Variable receiving result = v2ElemText)
XML Attribute Get (Element handle variable = "%v2ElemChild1%", Attribute name = "a1", Variable for result =
"v2A1AttributeValue")
XML Navigate to Element (Element handle variable = "v2XmlDoc", Path = "root\elem1", Element handle
variable = "v2ElemNavigated")
IF STRING %v2ElemNavigated%==%_vStrEmpty%
    Message SHOW "Information" : "Xml file open failed." (other parameters: x = -100, y = -100, Window title =
    Error, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
ENDIF
XML File Close "v2XmlDoc"
```

Example (Plain Text):

```
<#>This macro shows how to work with Xml file content
<#>Create XML file with some simple content and save it to disk
<cmds>
<xml_file_create>(vXmlDoc,vXmlRoot)
<xml_element_create>(vXmlDoc,elem1,This is elem1 text,vElem1)
<xml_attribute_set>(vElem1,a1,Hello)
<xml_element_set>(vXmlRoot,CHILD_FIRST,vElem1)
<xml_file_save>(vXmlDoc,"%TEMP%\test.xml")
<xml_file_close>(vXmlDoc)<#>Open the XML file previously created and explore the contet
<xml_file_open>("%TEMP%\test.xml",v2XmlDoc,v2XmlRoot)
<if_str>("%v2XmlDoc%==%_vStrEmpty%")
<msg>(-100,-100,"Xml file open failed.","Error",1,0,0,0)
<endif>
<xml_element_get>(v2XmlRoot,CHILD_FIRST,v2ElemChild1)
<if_str>("%v2ElemChild1%==%_vStrEmpty%")
<msg>(-100,-100,"Xml element not found.","Error",1,0,0,0)
<endif>
<xml_element_get>(v2ElemChild1,NAME,v2ElemName)
<xml_element_get>(v2ElemChild1,TEXT,v2ElemText)
<xml_attribute_get>(%v2ElemChild1%,a1,v2A1AttributeValue)
<xml_element_navigate>(v2XmlDoc,"root\elem1",v2ElemNavigated)
<if_str>("%v2ElemNavigated%==%_vStrEmpty%")
<msg>(-100,-100,"Xml file open failed.","Error",1,0,0,0)
<endif>
<xml_file_close>(v2XmlDoc)
```

File Close - < xml_file_close >() ... [Pro]

XML File Close
<xml_file_close>(File handle variable)
Available in: Professional edition

This command closes an opened (or created) Xml file.

#	Parameter name	Parameter description
1	File handle variable	Xml file handle - an Xml file identifier retrieved from "open xml" or "create xml" commands.

Example (Macro Steps):

1 <#> <#>This macro shows how to close Xml file

2 Macro execution: ONLY COMMANDS

3 XML File Create vXmlDoc, Root element handle variable = vXmlRoot,Password =

4 XML File Save "vXmlDoc" (File path = "c:\temp\newFile.xml")

5 XML File Close "vXmlDoc"

Example (Plain Text):

```
<#>This macro shows how to close Xml file
<cmds>
<xml_file_create>(vXmlDoc,vXmlRoot)
<xml_file_save>(vXmlDoc,"c:\temp\newFile.xml")
<xml_file_close>(vXmlDoc)
```

Navigate to Element - < xml_element_navigate >() ... [Pro]

XML Navigate to Element

<xml_element_navigate>(Element handle variable,"Path",Element handle variable)
Available in: Professional edition

This command navigates to an Xml file element.

#	Parameter name	Parameter description
1	Element handle variable	Starting element handle - variable that identifies the element where the path starts.
2	Path	Path to element to navigate to in form "element2\element3\...\elementX" where "element2" is the first child of the starting elemenet, "elemenet3" is the first child of "element2", etc.
3	Element handle variable	Name of variable that will receive handle of the element from the path ("elementX").

Example (Macro Steps):

```
<#> <#>This macro shows how to work with Xml file content
<#> <#>Create XML file with some simple content and save it to disk
Macro execution: ONLY COMMANDS
XML File Create vXmlDoc, Root element handle variable = vXmlRoot,Password =
XML Element Create (File handle variable = "vXmlDoc", Name = "elem1", Text = "This is elem1 text", Element handle variable = "vElem1")
XML Attribute Set (Element handle variable = "vElem1", Attribute name = "a1", Value = "Hello")
XML Element Set "vXmlRoot" (What = CHILD_FIRST, Input = vElem1)
XML File Save "vXmlDoc" (File path = "%TEMP%\test.xml")
XML File Close "vXmlDoc"
<#> <#>Open the XML file previously created and explore the contet
XML File Open File path = "%TEMP%\test.xml", File handle variable = "v2XmlDoc", Root element handle variable = "v2XmlRoot", Password = ""
IF STRING %v2XmlDoc%==%_vStrEmpty%
    Message SHOW "Information" : "Xml file open failed." (other parameters: x = -100, y = -100, Window title = Error, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
ENDIF
XML Element Get "v2XmlRoot" (What = CHILD_FIRST, Variable receiving result = v2ElemChild1)
IF STRING %v2ElemChild1%==%_vStrEmpty%
    Message SHOW "Information" : "Xml element not found." (other parameters: x = -100, y = -100, Window title = Error, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
ENDIF
XML Element Get "v2ElemChild1" (What = NAME, Variable receiving result = v2ElemName)
XML Element Get "v2ElemChild1" (What = TEXT, Variable receiving result = v2ElemText)
XML Attribute Get (Element handle variable = "%v2ElemChild1%", Attribute name = "a1", Variable for result = "v2A1AttributeValue")
XML Navigate to Element (Element handle variable = "%v2XmlRoot%", Path = "elem1", Element handle variable = "v2ElemNavigated")
XML Attribute Get (Element handle variable = "%v2ElemNavigated%", Attribute name = "a1", Variable for result = "vNavigatedValue")
IF STRING %vNavigatedValue%!=Hello
    Message SHOW "Information" : "Navigation to element failed. %vNavigatedValue%" (other parameters: x = -100, y = -100, Window title = Error, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
ENDIF
```

Example (Plain Text):

```
<#>This macro shows how to work with Xml file content
<#>Create XML file with some simple content and save it to disk
<cmds>
<xml_file_create>(vXmlDoc,vXmlRoot)
<xml_element_create>(vXmlDoc,elem1,This is elem1 text,vElem1)
<xml_attribute_set>(vElem1,a1,Hello)
<xml_element_set>(vXmlRoot,CHILD_FIRST,vElem1)
<xml_file_save>(vXmlDoc,"%TEMP%\test.xml")
<xml_file_close>(vXmlDoc)<#>Open the XML file previously created and explore the contet
<xml_file_open>("%TEMP%\test.xml",v2XmlDoc,v2XmlRoot)
<if_str>("%v2XmlDoc%==%_vStrEmpty%")
<msg>(-100,-100,"Xml file open failed.","Error",1,0,0,0)
<endif>
<xml_element_get>(v2XmlRoot,CHILD_FIRST,v2ElemChild1)
<if_str>("%v2ElemChild1%==%_vStrEmpty%")
<msg>(-100,-100,"Xml element not found.","Error",1,0,0,0)
<endif>
<xml_element_get>(v2ElemChild1,NAME,v2ElemName)
<xml_element_get>(v2ElemChild1,TEXT,v2ElemText)
<xml_attribute_get>(%v2ElemChild1%,a1,v2A1AttributeValue)
<xml_element_navigate>(%v2XmlRoot%,"elem1",v2ElemNavigated)
<xml_attribute_get>(%v2ElemNavigated%,a1,vNavigatedValue)
<if_str>("%vNavigatedValue%!=Hello")
<msg>(-100,-100,"Navigation to element failed.
%vNavigatedValue%","Error",1,0,0,0)
<endif>
<xml_file_close>(v2XmlDoc)
```

File Create - < xml_file_create >() ... [Pro]

XML File Create
<xml_file_create>(File handle variable,Root element handle variable,"Password")
Available in: Professional edition

This command creates a new Xml file. A "xml file save" command must be used in order to persist the Xml file on disk (or other media).The command returns file handle and root element handle (to variables passed as parameters) needed for other operarions.

#	Parameter name	Parameter description
1	File handle variable	Xml file handle - a unique number that is used as parameter to other commands (for eaxample, to save or close the Xml file).
2	Root element handle variable	Xml root element handle - a unique number that identifies the root element of the Xml file. It is used as input to other commands (for example, to get set child elements).
3	Password	Password used to encrypt the data. The password is used to encrypt the data that are set in XML file using 'xml_element_create', 'xml_element_set', and 'xml_attribute_set' commands. Leave the parameter empty (default value) if no encryption is required.

Example (Macro Steps):

```
1    <#> <#>This macro shows how to create a new empty Xml file
2    Macro execution: ONLY COMMANDS
3    XML File Create vXmlDoc, Root element handle variable = vXmlRoot,Password =
4    XML File Save "vXmlDoc" (File path = "c:\temp\newFile.xml")
5    XML File Close "vXmlDoc"
```

Example (Plain Text):

```
<#>This macro shows how to create a new empty Xml file
<cmds>
<xml_file_create>(vXmlDoc,vXmlRoot)
<xml_file_save>(vXmlDoc,"c:\temp\newFile.xml")
<xml_file_close>(vXmlDoc)
```

Element Set - < xml_element_set >() ... [Pro]

XML Element Set

<xml_element_set>(Element handle variable,What,Input)
Available in: Professional edition

This command allows to set some element properties such as element name or text as well as it allows to add a new child element.

#	Parameter name	Parameter description
1	Element handle variable	Variable that identifies the element.
2	What	What property to set: TEXT - element text defined by "Input" parameter is set. NAME - element name defined by "Input" parameter is set. CHILD_FIRST - element handle provided in "Input" parameter is set as the first child element. CHILD_LAST - element handle provided in "Input" parameter is set as the last child element.
3	Input	The meaning of this parameter dependes on the previous parameter.

Example (Macro Steps):

```
<#> <#>This macro shows how to work with Xml file content
<#> <#>Create XML file with some simple content and save it to disk
Macro execution: ONLY COMMANDS
XML File Create vXmlDoc, Root element handle variable = vXmlRoot,Password =
XML Element Create (File handle variable = "vXmlDoc", Name = "elem1", Text = "This is elem1 text",
Element handle variable = "vElem1")
XML Attribute Set (Element handle variable = "vElem1", Attribute name = "a1", Value = "Hello")
XML Element Set "vXmlRoot" (What = CHILD_FIRST, Input = vElem1)
XML File Save "vXmlDoc" (File path = "%TEMP%\test.xml")
XML File Close "vXmlDoc"
<#> <#>Open the XML file previously created and explore the contet
XML File Open File path = "%TEMP%\test.xml", File handle variable = "v2XmlDoc", Root element handle
variable = "v2XmlRoot", Password = ""
IF STRING %v2XmlDoc%==%_vStrEmpty%
    Message SHOW "Information" : "Xml file open failed." (other parameters: x = -100, y = -100, Window title =
    Error, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
ENDIF
XML Element Get "v2XmlRoot" (What = CHILD_FIRST, Variable receiving result = v2ElemChild1)
IF STRING %v2ElemChild1%==%_vStrEmpty%
    Message SHOW "Information" : "Xml element not found." (other parameters: x = -100, y = -100, Window
    title = Error, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
ENDIF
XML Element Get "v2ElemChild1" (What = NAME, Variable receiving result = v2ElemName)
XML Element Get "v2ElemChild1" (What = TEXT, Variable receiving result = v2ElemText)
XML Attribute Get (Element handle variable = "%v2ElemChild1%", Attribute name = "a1", Variable for result =
"v2A1AttributeValue")
XML Navigate to Element (Element handle variable = "v2XmlDoc", Path = "root\elem1", Element handle
variable = "v2ElemNavigated")
IF STRING %v2ElemNavigated%==%_vStrEmpty%
    Message SHOW "Information" : "Xml file open failed." (other parameters: x = -100, y = -100, Window title =
    Error, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
ENDIF
XML File Close "v2XmlDoc"
```

Example (Plain Text):

```
<#>This macro shows how to work with Xml file content
<#>Create XML file with some simple content and save it to disk
<cmds>
<xml_file_create>(vXmlDoc,vXmlRoot)
<xml_element_create>(vXmlDoc,elem1,This is elem1 text,vElem1)
<xml_attribute_set>(vElem1,a1,Hello)
<xml_element_set>(vXmlRoot,CHILD_FIRST,vElem1)
<xml_file_save>(vXmlDoc,"%TEMP%\test.xml")
<xml_file_close>(vXmlDoc)<#>Open the XML file previously created and explore the contet
<xml_file_open>("%TEMP%\test.xml",v2XmlDoc,v2XmlRoot)
<if_str>("%v2XmlDoc%==%_vStrEmpty%")
<msg>(-100,-100,"Xml file open failed.","Error",1,0,0,0)
<endif>
<xml_element_get>(v2XmlRoot,CHILD_FIRST,v2ElemChild1)
<if_str>("%v2ElemChild1%==%_vStrEmpty%")
<msg>(-100,-100,"Xml element not found.","Error",1,0,0,0)
<endif>
<xml_element_get>(v2ElemChild1,NAME,v2ElemName)
<xml_element_get>(v2ElemChild1,TEXT,v2ElemText)
<xml_attribute_get>(%v2ElemChild1%,a1,v2A1AttributeValue)
<xml_element_navigate>(v2XmlDoc,"root\elem1",v2ElemNavigated)
<if_str>("%v2ElemNavigated%==%_vStrEmpty%")
<msg>(-100,-100,"Xml file open failed.","Error",1,0,0,0)
<endif>
<xml_file_close>(v2XmlDoc)
```

Attribute Set - < xml_attribute_set >() ... [Pro]

XML Attribute Set
<xml_attribute_set>(Element handle variable,Attribute name,Value)
Available in: Professional edition

This command allows to set an attribute value for given Xml file element.

#	Parameter name	Parameter description
1	Element handle variable	Variable that identifies the element.
2	Attribute name	Attribute name. If the attribute does not exist then it is added.
3	Value	value to set.

Example (Macro Steps):

```
<#> <#>This macro shows how to work with Xml file content
<#> <#>Create XML file with some simple content and save it to disk
Macro execution: ONLY COMMANDS
XML File Create vXmlDoc, Root element handle variable = vXmlRoot,Password =
XML Element Create (File handle variable = "vXmlDoc", Name = "elem1", Text = "This is elem1 text",
Element handle variable = "vElem1")
XML Attribute Set (Element handle variable = "vElem1", Attribute name = "a1", Value = "Hello")
XML Element Set "vXmlRoot" (What = CHILD_FIRST, Input = vElem1)
XML File Save "vXmlDoc" (File path = "%TEMP%\test.xml")
XML File Close "vXmlDoc"
<#> <#>Open the XML file previously created and explore the contet
XML File Open File path = "%TEMP%\test.xml", File handle variable = "v2XmlDoc", Root element handle
variable = "v2XmlRoot", Password = ""
IF STRING %v2XmlDoc%==%_vStrEmpty%
    Message SHOW "Information" : "Xml file open failed." (other parameters: x = -100, y = -100, Window title =
    Error, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
ENDIF
XML Element Get "v2XmlRoot" (What = CHILD_FIRST, Variable receiving result = v2ElemChild1)
IF STRING %v2ElemChild1%==%_vStrEmpty%
    Message SHOW "Information" : "Xml element not found." (other parameters: x = -100, y = -100, Window
    title = Error, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
ENDIF
XML Element Get "v2ElemChild1" (What = NAME, Variable receiving result = v2ElemName)
XML Element Get "v2ElemChild1" (What = TEXT, Variable receiving result = v2ElemText)
XML Attribute Get (Element handle variable = "%v2ElemChild1%", Attribute name = "a1", Variable for result =
"v2A1AttributeValue")
XML Navigate to Element (Element handle variable = "v2XmlDoc", Path = "root\elem1", Element handle
variable = "v2ElemNavigated")
IF STRING %v2ElemNavigated%==%_vStrEmpty%
    Message SHOW "Information" : "Xml file open failed." (other parameters: x = -100, y = -100, Window title =
    Error, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
ENDIF
XML File Close "v2XmlDoc"
```

Example (Plain Text):

```
<#>This macro shows how to work with Xml file content
<#>Create XML file with some simple content and save it to disk
<cmds>
<xml_file_create>(vXmlDoc,vXmlRoot)
<xml_element_create>(vXmlDoc,elem1,This is elem1 text,vElem1)
<xml_attribute_set>(vElem1,a1,Hello)
<xml_element_set>(vXmlRoot,CHILD_FIRST,vElem1)
<xml_file_save>(vXmlDoc,"%TEMP%\test.xml")
<xml_file_close>(vXmlDoc)<#>Open the XML file previously created and explore the contet
<xml_file_open>("%TEMP%\test.xml",v2XmlDoc,v2XmlRoot)
<if_str>("%v2XmlDoc%==%_vStrEmpty%")
<msg>(-100,-100,"Xml file open failed.","Error",1,0,0,0)
<endif>
<xml_element_get>(v2XmlRoot,CHILD_FIRST,v2ElemChild1)
<if_str>("%v2ElemChild1%==%_vStrEmpty%")
<msg>(-100,-100,"Xml element not found.","Error",1,0,0,0)
<endif>
<xml_element_get>(v2ElemChild1,NAME,v2ElemName)
<xml_element_get>(v2ElemChild1,TEXT,v2ElemText)
<xml_attribute_get>(%v2ElemChild1%,a1,v2A1AttributeValue)
<xml_element_navigate>(v2XmlDoc,"root\elem1",v2ElemNavigated)
<if_str>("%v2ElemNavigated%==%_vStrEmpty%")
<msg>(-100,-100,"Xml file open failed.","Error",1,0,0,0)
<endif>
<xml_file_close>(v2XmlDoc)
```

Element Create - < xml_element_create >() ... [Pro]

XML Element Create

<xml_element_create>(File handle variable,Name,Text,Element handle variable)
Available in: Professional edition

This command creates a new Xml file element. Once the new element is created it can be added to Xml file using "element set" command.

#	Parameter name	Parameter description
1	File handle variable	Xml file handle - an Xml file identifier provided from "open xml" or "create xml" commands.
2	Name	Name of the new element.
3	Text	The new element text.
4	Element handle variable	Name of variable that will receive the newly created element handle.

Example (Macro Steps):

```
<#> <#>This macro shows how to work with Xml file content
<#> <#>Create XML file with some simple content and save it to disk
Macro execution: ONLY COMMANDS
XML File Create vXmlDoc, Root element handle variable = vXmlRoot,Password =
XML Element Create (File handle variable = "vXmlDoc", Name = "elem1", Text = "This is elem1 text",
Element handle variable = "vElem1")
XML Attribute Set (Element handle variable = "vElem1", Attribute name = "a1", Value = "Hello")
XML Element Set "vXmlRoot" (What = CHILD_FIRST, Input = vElem1)
XML File Save "vXmlDoc" (File path = "%TEMP%\test.xml")
XML File Close "vXmlDoc"
<#> <#>Open the XML file previously created and explore the contet
XML File Open File path = "%TEMP%\test.xml", File handle variable = "v2XmlDoc", Root element handle
variable = "v2XmlRoot", Password = ""
IF STRING %v2XmlDoc%==%_vStrEmpty%
    Message SHOW "Information" : "Xml file open failed." (other parameters: x = -100, y = -100, Window title =
    Error, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
ENDIF
XML Element Get "v2XmlRoot" (What = CHILD_FIRST, Variable receiving result = v2ElemChild1)
IF STRING %v2ElemChild1%==%_vStrEmpty%
    Message SHOW "Information" : "Xml element not found." (other parameters: x = -100, y = -100, Window
    title = Error, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
ENDIF
XML Element Get "v2ElemChild1" (What = NAME, Variable receiving result = v2ElemName)
XML Element Get "v2ElemChild1" (What = TEXT, Variable receiving result = v2ElemText)
XML Attribute Get (Element handle variable = "%v2ElemChild1%", Attribute name = "a1", Variable for result =
"v2A1AttributeValue")
XML Navigate to Element (Element handle variable = "v2XmlDoc", Path = "root\elem1", Element handle
variable = "v2ElemNavigated")
IF STRING %v2ElemNavigated%==%_vStrEmpty%
    Message SHOW "Information" : "Xml file open failed." (other parameters: x = -100, y = -100, Window title =
    Error, Buttons = OK, Timeout (seconds) = 0, Always on top = No).
ENDIF
XML File Close "v2XmlDoc"
```

Example (Plain Text):

```
<#>This macro shows how to work with Xml file content
<#>Create XML file with some simple content and save it to disk
<cmds>
<xml_file_create>(vXmlDoc,vXmlRoot)
<xml_element_create>(vXmlDoc,elem1,This is elem1 text,vElem1)
<xml_attribute_set>(vElem1,a1,Hello)
<xml_element_set>(vXmlRoot,CHILD_FIRST,vElem1)
<xml_file_save>(vXmlDoc,"%TEMP%\test.xml")
<xml_file_close>(vXmlDoc)<#>Open the XML file previously created and explore the contet
<xml_file_open>("%TEMP%\test.xml",v2XmlDoc,v2XmlRoot)
<if_str>("%v2XmlDoc%==%_vStrEmpty%")
<msg>(-100,-100,"Xml file open failed.","Error",1,0,0,0)
<endif>
<xml_element_get>(v2XmlRoot,CHILD_FIRST,v2ElemChild1)
<if_str>("%v2ElemChild1%==%_vStrEmpty%")
<msg>(-100,-100,"Xml element not found.","Error",1,0,0,0)
<endif>
<xml_element_get>(v2ElemChild1,NAME,v2ElemName)
<xml_element_get>(v2ElemChild1,TEXT,v2ElemText)
<xml_attribute_get>(%v2ElemChild1%,a1,v2A1AttributeValue)
<xml_element_navigate>(v2XmlDoc,"root\elem1",v2ElemNavigated)
<if_str>("%v2ElemNavigated%==%_vStrEmpty%")
<msg>(-100,-100,"Xml file open failed.","Error",1,0,0,0)
<endif>
<xml_file_close>(v2XmlDoc)
```

Find Text - < xml_findtext >() ... [Pro]

XML Find Text

<xml_findtext>(Start element handle,"Element name","Attribute name","Text to find",Variable for matching elements,Variable for number of matching elements)
Available in: Professional edition

This command finds text in Xml document and provides list of elements where the text was found.

#	Parameter name	Parameter description
1	Start element handle	The Xml element handle to start searching from. This can be typically the root element of the Xml document.
2	Element name	The name of the element to search for the text in. For example, "address" for the <address> element. If this field is empty then all elements are searched. Wildcards such as * or ? can be used.
3	Attribute name	The name of the attribute to search for the text in. For example, "style" for the <p style="text-align:right;> element. If this field is empty then all attributes are searched. Wildcards such as * or ? can be used.
4	Text to find	The text to find. The text is searched in both element text values and attributes' values. Wildcards such as * or ? can be used.
5	Variable for matching elements	The name of variable (variable array) that receives handles of all elements where the text was found. For example, vFoundInElements.
6	Variable for number of matching elements	The name of variable that receives the number of matching elements. For example, vFoundInElementsCount.

Example (Macro Steps):

```
1   <#> <#>This macro shows how to use the "xml find element" command to retrieve the latest version
    fo the Macro Toolworks from the web
2   Macro execution: ONLY COMMANDS
3   <#> <#> Download the Macro Toolworks page
4   Http DOWNLOAD "http://www.macrotoolworks.com" (Login name=) to file "c:\temp\macrotoolworks.html"
5   <#> <#> Convert it to Xml document
6   File Convert HTML to XML Convert HTML file c:\temp\macrotoolworks.html to XML file
    c:\temp\macrotoolworks.xml
7   XML File Open File path = "c:\temp\macrotoolworks.xml", File handle variable = "vA01", Root element handle
    variable = "vA01Root", Password = ""
8   <#> <#> Find the version in the document
9   XML Find Text "Version*" (Element name=*, Attribute name=*, Variable for matching elements =
    vElemsFound, Variable for number of matching elements = vElemsFoundCount)
10  IF NUMERIC %vElemsFoundCount%>0
11      XML Element Get "%vElemsFound[0]%" (What = TEXT, Variable receiving result = vVersion)
12      Message SHOW "Information" : "This is the latest version available: %vVersion%" (other parameters: x =
        -100, y = -100, Window title = Macro Toolworks Version Info, Buttons = OK, Timeout (seconds) = 0,
        Always on top = No).
13      Macro EXIT
14  ENDIF
15  Message SHOW "Error" : "Not able to retrieve the version information." (other parameters: x = -100, y = -100,
    Window title = , Buttons = OK, Timeout (seconds) = 0, Always on top = No).
```

Example (Plain Text):

```
<#>This macro shows how to use the "xml find element" command to retrieve the latest version fo the Macro Toolworks
from the web
<cmds>
<#> Download the Macro Toolworks page
<download>("c:\temp\macrotoolworks.html","http://www.macrotoolworks.com","","")
<#> Convert it to Xml document
<file_html2xml>("c:\temp\macrotoolworks.html","c:\temp\macrotoolworks.xml")
<xml_file_open>("c:\temp\macrotoolworks.xml",vA01,vA01Root)
<#> Find the version in the document
<xml_findtext>(vA01Root,"*","*","Version*",vElemsFound,vElemsFoundCount)
<if_num>("%vElemsFoundCount%>0")
<xml_element_get>(%vElemsFound[0]%,TEXT,vVersion)
<msg>(-100,-100,"This is the latest version available:

%vVersion%","Macro Toolworks Version Info",1,0,0,0)
<exitmacro>
<endif>
<msg>(-100,-100,"Not able to retrieve the version information.","",1,0,2,0)
```

How To Write Reliable Macros?

How To Write Reliable Macros

It often happens that a macro (macro that sends keystrokes or uses mouse events) user manually creates or records works fine at the development time but starts being unreliable later (after user reboots computer, installs new software etc.). This is not a problem of the macro program but it is natural problem of Windows events timing. One simple example:
Let's say user wants to create macro that will do this: Start Notepad and type "Hello" in it. This is quite simple and user quickly creates this macro:

<lwinkey>r<lwinkey>notepad.exe<enter>Hello

The macro will do this: opens Run dialog box (<lwinkey>r<lwinkey>), types "Notepad.exe" in it, hits Enter key to run Notepad and types "Hello" in it. In 90% cases, It will work OK. But what if: (i) computer is busy and Run dialog appears many seconds after all the keystrokes (Notepad.exe Hello) are already sent out, (ii) a newly installed software user re-defines LWinKey+R hot key and Run dialog doesn't appear at all. In such cases the macro fails doing what's expected without any notice to the user. The macro language has commands that make it possible to write the same macro safe way:

```
<lwinkey>r<lwinkey><cmds> <#> Open Run dialog

<waitfor>("WIN","ACT","Run",5,0) <#> Wait for the dialog to become active
<if_str>("_vErr==NO")
  <keys>notepad.exe<newline><cmds> <#> No (timeout) error? Let the dialog to start Notepad then...
<else>
  <msg>(-100,-100,"%_vQuoteChar%Run%_vQuoteChar% dialog faild to open.","Message",1)
  <exitmacro>
<endif>

<waitfor>("WIN","ACT","Notepad",5,0) <#> Wait for Notepad to become active
<if_str>("_vErr==NO")
  <keys>Hello<cmds> <#> No (timeout) error? Send keystrokes to Notepad then...
<else>
  <msg>(-100,-100,"%_vQuoteChar%Notepad%_vQuoteChar% not activated.","Message",1)
  <exitmacro>
<endif>
```

This code is much longer compare to the original macro, right. But the macro execution result is always deterministic: Macro either executes OK or any failure is properly handled. This becomes important especially for more complex macros that many users use.

There are a few advises that should help you to write macros that are more reliable:

1. Some time an application the macro runs in is too slow to process all the input (keystrokes sent) in time. In such case it can help to use **<wx>(250)** command that stops macro execution for 250ms and gives the application time to process previous input before the macro continues.

2. If you write a macro that starts an application and then sends keystrokes into it, it is necessary to wait until the application loads. You can use **<waitfor>** command right after the command you use to start the application. The <waitfor> command waits until specified window appears on the screen or until it times out (timeout sets *_vErr* system variable and thus can be handled in the macro code).

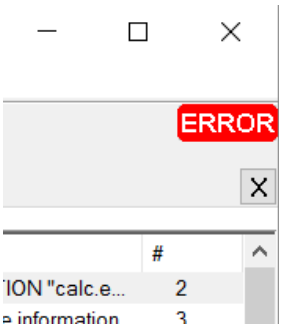
3. If you write a macro that copies some selected data (text, bitmap, etc.) using clipboard, it is recommended to use <clp_copyselected> command (this command waits until data are really copied to clipboard before macro execution continues) or use <waitfor> command to wait until the data are actually saved in the clipboard.

4. Replace all keystrokes and mouse actions by other commands if possible. This means that if something can be done using a macro language command instead of using keyboard or mouse then always use the command. For example, people often tend to open applications by simulating clicks on desktop or Start menu. This has many drawbacks and it is always better to use <execappex> command. We can simplify and make our sample above more robust using this command:

```
<cmds>
 <execappex>("Notepad.exe","","",0,0)
 <waitfor>("WIN","ACT","Notepad",5,0)
 <if_str>("_vErr==NO")
   <keys>Hello<cmds>
 <else>
   <msg>(-100,-100,"%_vQuoteChar%Notepad%_vQuoteChar% not activated.","Message",1)
   <exitmacro>
 <endif>
```

Troubleshooting

If error is indicated in the main window upper right area then some triggers may not work properly. Try this to fix it:



1. Restart Macro Toolworks.
2. Reboot computer.
3. Check your anti-virus or anti-logger software and make sure that Macro Toolworks is not block. Eventually, add Macro Toolworks to white list.
4. Make sure that Macro Toolworks has write access to folder were it is installed (write access to "Program Files" folder is sometimes set so that only processes with Administrator rights can write to the folder).
5. Reinstall Macro Toolworks. Eventually install the program to new location - see #4 above.

Index