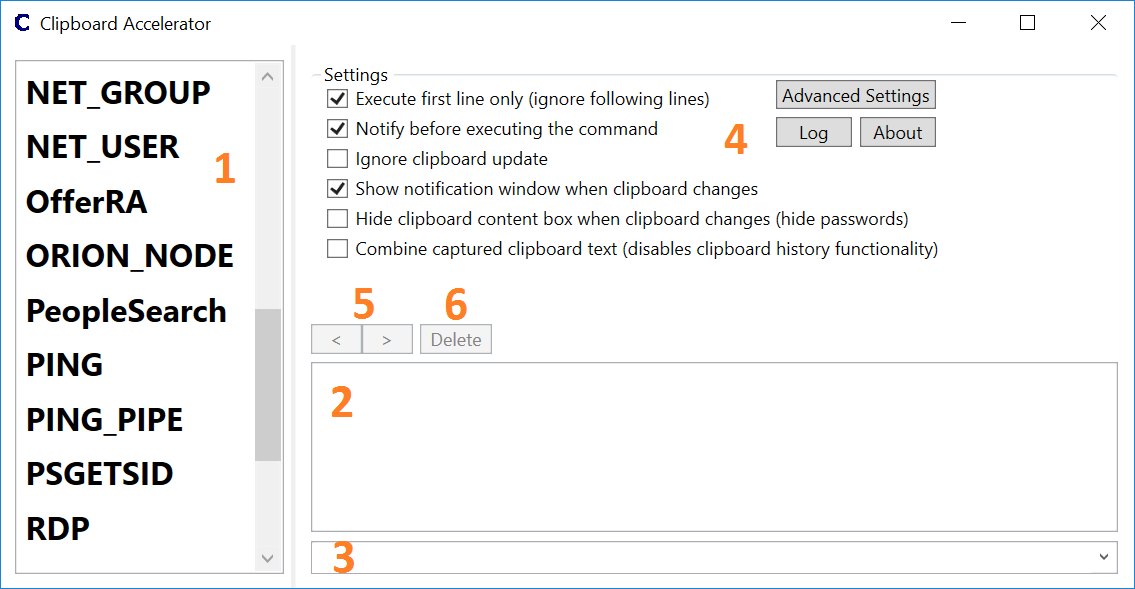
# **Clipboard Accelerator**



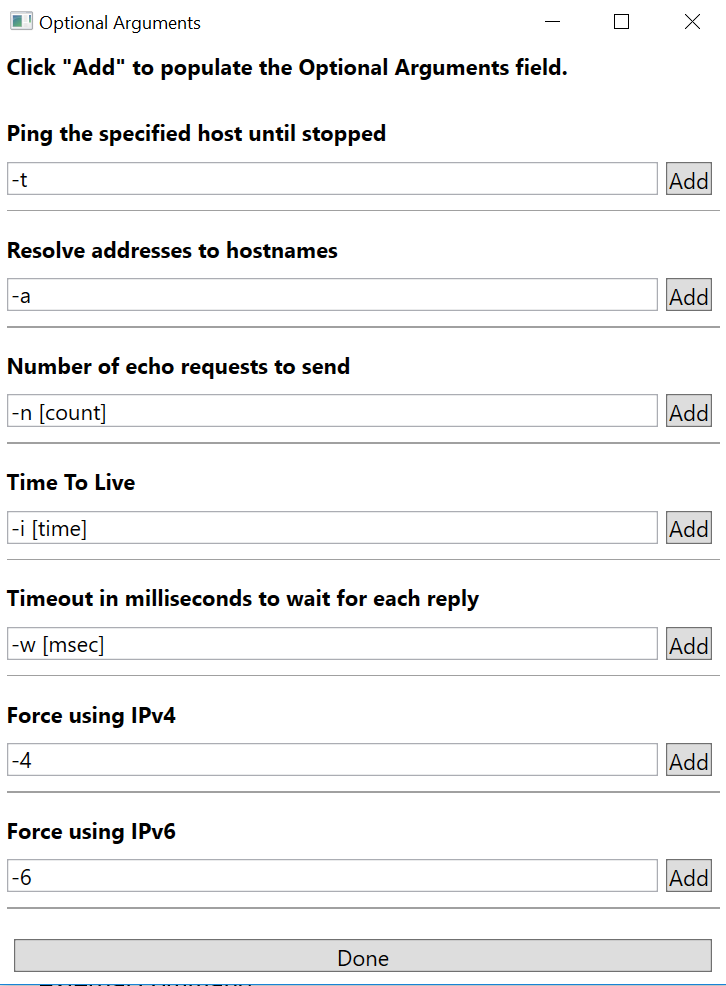
# Main Functionality

1. List of external commands
   1. The external commands can be one of the following file formats: .BAT, .CMD, .PS1 or XML. The tool will automatically list such files in the list box (1) when placed into the “Tools” directory which needs to be in the directory where Clipboard Accelerator was started from.
   2. Double clicking an entry in the list of external commands will run the external command with the text in textbox (2) as an argument. Named pipes to pass the text to the external command is also supported (with XML files only).
   3. A right click to an external command will show a context menu with the entry “Optional Arguments”. This can be used to select pre-define optional arguments which can be passed to the external command. (“Optional Arguments” => to be documented)
2. Textbox containing the arguments
   1. The textbox contains either text entered manually or text which was put to the clipboard and captured by the tool.
   2. Each single line (except empty lines) will be treated as an individual argument to an external command.  
      External commands defined in XML files can use named pipes (to be documented) to communicate with the clipboard accelerator tool. In such a case the whole text in the textbox will be passed to the external command using the named pipe. The external command can handle the received text data as required.
3. Optional arguments drop down list / input field
   1. This can be used to pass optional arguments to the external command. The optional arguments can be entered manually or populated using the optional arguments functionality described in 1c above. This is only supported with external commands defined in XML files.
4. Settings section
   1. Execute first line only (ignore following lines)
      1. If this checkbox is checked only the first line in the textbox (2) will be passed to the selected external command. This means, that the external command will be executed only once with the first line in the textbox (2).
      2. If this checkbox is cleared all lines will be passed as single arguments to the external command. This causes the external command to be executed multiple times. One time for each line in the textbox (2). In case the external command is defined in an XML file and the “pipe” option is set, it uses a named pipe to receive the whole textbox content at once. In such a case the external command is triggered only once.
   2. Notify before executing the command
      1. If this checkbox is checked a confirmation window is shown before executing the external command.
      2. If it is cleared the external command is executed immediately with no confirmation request. *This might be dangerous.*
      3. The setting of this checkbox can be overridden when using an XML file to define the external command. Details can be found in the section describing the structure of the XML files.
   3. Ignore clipboard updates
      1. If this checkbox is checked the tool will not capture text on the clipboard.
   4. Show notification window when clipboard changes
      1. If this checkbox is checked a notification window will be shown when new text was captured from the clipboard. Move the cursor over this notification window to show Clipboard Accelerator.  
         
   5. Hide clipboard content box when clipboard changes (hide passwords)
      1. If this checkbox is checked the clipboard textbox is hidden behind a button. The captured clipboard content is shown again when the button is clicked. This functionality can be used to prevent the unintended leakage of copied passwords.  
         *Make sure to delete passwords form the list of captured clipboards as soon as possible.*
   6. Combine captured clipboard text (disables clipboard history functionality)
      1. If this checkbox is checked all new captured text from the clipboard will be added to the last clipboard text shown in the tool. The tool always adds the new copied text after a line break.  
         *This functionality disables the clipboard history functionality.*
5. Go back and forth in the list of captured clipboards
6. Delete the currently visible captured clipboard from the saved clipboard list

# Advanced Settings

1. Re-enable option “Execute first line only” after executing multiple lines
   1. If this checkbox is checked the tool will automatically check the checkbox “Execute first line only (ignore following lines)” after an external command has been executed. This ensures that external commands are not executed multiple times unintentionally.
2. Hide from taskbar when minimized
   1. If this checkbox is checked the tool will hide its taskbar icon when the minimize button in the window title is clicked.
3. Line count to trigger execution warning
   1. If executing an external command multiple times (or using the pipe with multiple lines) this setting defines at how many lines the tool will show a warning message to prevent executing the command multiple times unintentionally.
4. Copy N bytes from the clipboard
   1. This setting defines how many bytes the tool should copy from the clipboard. Since the tool will save the copied clipboard data for later use it could consume lots of memory if it would copy all data. The second reason for this setting is the textbox showing the clipboard content. The textbox performs very poorly if there is lots of data to show. The default value for this setting is 500000 bytes.
5. Clipboard access delay in milliseconds
   1. Defines how long the tool waits to access the clipboard the next time. This is required since some programs (e.g. MS Excel) put data to the clipboard in multiple formats (e.g. raw text, html, ...) which causes multiple clipboard updates within a very short period of time. In such a case the Clipboard Accelerator tool would see multiple clipboard update events and may show “false” clipboard updates. Furthermore, only one process can access the clipboard at the same time which could affect the clipboard update process of the other program which tries to update the clipboard content multiple times. The default value is 500 milliseconds.
6. Show the notification window for N milliseconds
   1. Defines how long the clipboard change notification window is shown.
7. Font size of the commands listbox
   1. This setting will set the font size of the commands listbox (1).
8. Notification window position
   1. Using this option the notification window can be placed at any location of the screen.

# Optional Arguments

1. If an external command supports multiple arguments those can be pre-configured in the XML file which points to the external command. For example the “ping.exe” command supports the argument “-t” to ping a target until interrupted. This argument could be added to the XML file to make it visible in the optional arguments window. When right clicking an external command in the external commands listbox (1) a context menu will open showing the option “Optional Arguments”. Selecting this will show a window with pre-defined optional arguments as shown in the below screenshot.  
     
   Clicking “Add” next to an argument will add it to the optional arguments drop down list in the main window. Usually “ping.exe” pings the destination 4 times. With a pre-defined optional argument, the “-t” command can be selected to ping the host until interrupted. Optional arguments can always be added manually (only supported with external commands defined using XML files) to the optional arguments drop down list in the main window. The pre-defined optional arguments are for convenience only.  
   The XML configuration files section below describes how to define optional arguments.

# Defining External Commands

Clipboard Accelerator can call external commands in three different ways.

1. BAT and CMD files
   * This file types are called with the below command line:  
     *CMD.EXE /K [path\_to\_CMD/BAT\_file] [line N of visible clipboard/text]*
   * The CMD/BAT file content layout should be:  
     *[command] %\**%\* will be replaced with the full argument, e.g. the line visible in the Clipboard Accelerator tool.
2. PS1 files
   * In this case Powershell.exe is called with the following parameters:  
     *-NoLogo -NoExit -ExecutionPolicy Bypass -File [path\_to\_PS1\_file] [line N of visible clipboard/text]*
3. XML files
   * XML files are not directly executed. The content is used to define which command is called in which way. An example for “ping.exe” is shown below:

|  |
| --- |
| *<?xml version="1.0" encoding="utf-8"?>*  *<ExternalProgram>*  *<program id="1">*  *<class>Computer</class>*  *<description>Ping a network device</description>*  *<issafe>true</issafe>*  *<usepipe>false</usepipe>*  *<path>C:\windows\system32</path>*  *<executable>cmd.exe</executable>*  *<staticarg>/K ping.exe %%oa\*\* %%ca\*\*</staticarg>*  *<option desc="Ping the specified host until stopped">-t</option>*  *<option desc="Resolve addresses to hostnames">-a</option>*  *<option desc="Milliseconds to wait for reply">-w [msec]</option>*  *</program>*  *</ExternalProgram>* |

In the above example cmd.exe is started with the parameter /K to keep the window open after the command “ping.exe” has completed.

# List of supported elements in an XML file defining external commands

|  |  |
| --- | --- |
| **XML element** | **Description** |
| ExternalProgram | Root element. This is mandatory. |
| Program | The child elements of the program element define how the external program should be executed. Currently only one program element is supported. It must be defined as shown below:  *<program id="1">* |
| Class | The class element is used to highlight the entry in the “list of external command” list box if text in the first line of the textbox (2) matches with a defined regular expression (see section “Defining regular expressions for command highlighting”). |
| Description | The description element is currently only used for logging purposes. |
| Issafe | The issafe element can be either “true” or “false” and defines if the option “Notify before executing the command” should be overridden. If set to “true”, Clipboard Accelerator will not notify before executing the command. Be careful with this setting. |
| Usepipe | The usepipe element specifies if the program will pass the parameters to the external program using a named pipe. See the PS1 file example how to access the pipe via PowerShell. |
| Path | The path element defines the directory path to the executable. |
| Executable | The executable element defines the external program which should be executed. |
| Staticarg | The staticarg element defines the command line argument which should be passed to the external program when starting it.  The following three placeholders can be embedded into the “staticarg” string:   * %%ca\*\* will be replaced with line N in the textbox containing the arguments (2) which is either the text captured from the clipboard or entered manually. * %%oa\*\* will be replaced with the optional argument specified in the optional arguments drop down list (3) of the tool. * %%pn\*\* will be replaced with the randomly generated named pipe name. |
| Option | The option element defines an optional argument and its description. This optional argument is shown in the “Optional Arguments” window described earlier. Multiple option elements are allowed.  An example to use the option with the “ping.exe” command would be:  <option desc="Ping until interrupted by user">-t</option> |
| ShellExecute | If the shellexecute option is set to “true” ClipboardAccelerator executes the string defined in staticarg with the shells (e.g. Explorer.exe) default handler.  It is similar to executing the command using the Explorer’s “Run” dialog box.  If shellexecute is set to “true” the following options are ignored:  Executable, Path, DllNamespaceName, DllClassName, DllMethodName, DllConfigFilePath |
| Isdll | The isdll option can be “true” or “false”. If it is true ClipboardAccelerator will load a .NET library / assembly instead of executing an external command. This functionality can be used to extend ClipboardAccelerator using .NET libraries. |
| DllNamespaceName | The namespace of the .NET dll / assembly to use. |
| DllClassName | The class name of the .NET dll / assembly to use. |
| DllMethodName | The method name to execute in the above class / namespace of the .NET dll / assembly. |
| DllConfigFilePath | This option can be used if the .NET dll / assembly requires a configuration file. It can hold a string which represents the path to the configuration file. |

# Defining regular expressions for command highlighting

The first line of the textbox containing the arguments (2) is used to match against regular expressions. These expressions can be defined in the file “RegEx.xml” located in the “Config” directory. The file layout is shown below:

|  |
| --- |
| *<?xml version="1.0" encoding="utf-8"?>*  *<RegExConfig>*  *<RegEx id="1">*  *<class>Computer</class>*  *<regexstring>^\s\*(\w{6}-\w{1,2}\d{4,6})(\s?|\r\n?|\n)?$</regexstring>*  *</RegEx>*  *<RegEx id="2">*  *<class>file</class> <regexstring>^\s\*(\w:\\(\w+\\)\*\w\*\.?\w{0,10})(\s?|\r\n?|\n)?$</regexstring>*  *</RegEx>*  *</RegExConfig>* |

All elements are mandatory for a working command highlighting. In the above example the first <RegEx> element defines a “Computer” class with a <regexstring> element containing the regular expression. If this expression matches the string in the first line of the textbox containing the arguments (2) all XML based entries in the external commands listbox (1) are highlighted which have the same text string in its <class> element.

# Example DLL / Assembly

An example can be found at <https://github.com/clepaul/ExampleDLL_for_ClipboardAccelerator>.

To create one from scratch using Visual Studio 2019 follow the below steps:

1. Crate new project in VS2019: WPF User Control Library (.NET Framework)
2. Select NET Framework 4.5.2
3. As mentioned at "https://stackoverflow.com/questions/3573339/no-creation-of-a-wpf-window-in-a-dll-project" change <UserControl> to <Window> in the Xaml file
4. Rename the class with the VS "rename" functionality if required
5. Add "string[] ClipboardText, XMLRecord xmlRecord" to the parameter list of the ctor of the new window
6. Add the "XMLRecord.cs" class from the example DLL and remove the logic which is not required or implement on your own
7. Add the "DllEntry.cs" file from the example DLL and update the window instance creation to the name of your window class (the name can be found in the first line of the xaml file)
   1. In the corresponding ".xaml.cs" file change the inherited class name from "UserControl" to "Window"
8. Change the output from "Debug" to "Release"
9. Add your code and compile

# To be documented

1. *Describe PS1 usage*
2. *Describe pipe usage*
3. *Create template PS1 file for pipe usage*
4. *Describe special character behavior (&, |, etc.)*