

# X-Launcher - The Universal Launcher!

1. [Introduction](#)
2. [Basic Concepts](#)
3. [Function References](#)
4. [Advanced Concepts](#)
5. [Appendix I – Major Environment Variables](#)
6. [Appendix II – AutoIt Macros](#)

# Introduction

## What

X-Launcher is a program that launches other programs. X-Launcher allows you to change at will the options for initiating programs undertaken in order to make them portable, that is usable on removable storage devices like USB keys or external hard disks.

X-Launcher is universal in the sense that it is more configurable and can be used to make portable applications without any limitations.

The idea is very simple: X-Launcher has a large number of tools to achieve its purpose, and all these instruments can be configured through a configuration file. You do not need to know any programming language to create a portable launcher that makes the desired program.

In summary it can be said that once the changes are clear to make a program portable, X-Launcher can execute them easily and effectively.

## Who Is

X-Launcher is a very versatile instrument, but rather difficult to use. The difficulty is mainly that we have to very clearly define what we must do to make a program portable, but everything else is simple!

To use X-Launcher it is not necessary to know specific programming languages, but a complete understanding of the program you want to make portable, and especially how to do so.

You must have a good familiarity with concepts such as:

- Command Line Options
- Environment Variables
- Registry Keys
- Configuration Files

## Portable Programs: How to

In general, a program is called “portable” if it can be run from a removable memory media (USB key, portable hard disk, etc...) and in this scenario works properly on any computer. In particular, this means that:

1. Its settings must be stored in the device and not in the host computer
2. All files necessary to run the program must reside in the device

Besides these two conditions (whose implementation is often not recent), we must keep in mind that often, in their operation, the applications refer to absolute paths, which are no longer valid when changing computers.

The main problem to solve is to force the program to save settings in your portable device and not in the host computer. From this point of view, you have these types of programs:

1. Programs that save the settings in %USERPROFILE% or in %APPDATA%
2. Programs that use the Windows registry. Generally using subkey: “HKEY\_CURRENT\_USER\Software”
3. Programs that save the settings in the program. These are “naturally” portable, although often require some changes.

Unfortunately, it is not always easy to understand how a program works and then modify it. In general, you can refer to resources supportive of the program, namely:

- User Manual
- Sites and Support Forums
- Sites specializing in portable programs

### **Programs that use %USERPROFILE% or %APPDATA%**

There are basically three techniques to force such programs to save user settings in a specific folder outside %USERPROFILE% or %APPDATA%:

1. Start the program using parameters that specify the location of the folder to be used for the settings. Examples of such programs are: Mozilla Firefox and Abakt.
2. Start the program by setting the path to the folder through user environment variables. Examples of such programs are: Amaya and Gaim.
3. Modify some configuration files. Examples of such programs are: ClamWin, OpenOffice, Opera.

### **Programs that use the Windows Registry**

In this case, the procedure is somewhat laborious, and consists of the steps described below.

Suppose that the program “MyApp” saves your settings in the registry key “HKCU\Software\MyApp. To use these settings on different computers, you must save it to a log file, which we will call “settings.reg”.

Suppose also that we are in a more complex situation, which is one where there is another copy of the same program “resident” on the same computer.

The operations will be carried out thus:

1. Backup key “HKCU\Software\MyApp”, with “resident” program settings in a log file, which we will call “backup.reg”
2. Delete key “HKCU\Software\MyApp”. This serves to initialize the working environment of our portable program.
3. Loading settings saved in the log file “settings.reg”
4. Startup and use of portable program
5. Closure of the portable program

6. Export key “HKCU\Software\MyApp” to the file “settings.reg
7. Delete key “HKCU\Software\MyApp”
8. Restore “resident” program settings previously saved in the file “backup.reg”

Of course, X-Launcher allows you to perform all of these operations completely automated. Examples of such programs are: Audacity, 7Zip, WackGet.

# Basic Concepts

## The configuration file \$ScriptIni\$

When you first start, X-Launcher creates the file “X-Launcher.ini” which will be configured to start the desired application. So knowing how to use X-Launcher means knowing how to configure this file.

The configuration file is located in the same folder as the launcher, and has the same name but with “.ini”. If you rename the executable, you will also rename it’s configuration file.

**From now on, the configuration file will be shown as: \$ScriptIni\$.**

Once l “.ini” is configured, you can recompile the launcher icon editing other parameters to create a custom launcher.

In an “.ini” file, instructions are organized in sections, like this:

```
[Section]
Key=Value
```

In regard to sections, we must emphasize that:

1. The names of the sections must be unique: **There cannot be two sections with the same name**
2. The order in which the sections are written in the “.ini” file have no importance

Within each section keys can be configured with their values. Depending on the sections, the order keys are written can be important. Also there may multiple sections containing keys with the same name but different values.

The following is the complete list of sections that you can configure, in the order in which they are interpreted by X-Launcher:

1. [Setup]
2. [FileSystem]
3. [FileToRun]
4. [Options]
5. [SplashScreen]
6. [Environment]
7. [Functions]
8. Sections that rewrite files: Executed in the order they appear in \$ScriptIni\$
  - o [StringReplace]
  - o [WriteToFile]
  - o [WriteToIni]
  - o [WriteToPref]
  - o [WriteToReg]
9. [RunBefore]
10. [RunAfter]

## Variables

The power of X-Launcher is in the variables. Three types of variables can be used:

- Internal Variables
- AutoIt Macros
- Environment Variables

### Internal Variables

These variables are internal to X-Launcher and are defined in the source code. The value of these variables can be defined directly in the configuration file, or dependant on the value of other variables.

To use them, you need to enclose the variable name between two \$ symbols:

1. **\$internal\_variable\$**

A list of the internal variables available to use are given in the Appendix.

### Macros

The AutoIt macro (the language in which X-Launcher is written) contains special variables that are internal programming language.

To use them, you need to enclose the name of the macro between two @ symbols:

1. **@autoit\_macro@**

The list of what macros can be used is found in the guidelines AutoIt, chapter: “Macro Reference”.

### Environment Variables

The environment variables (environment variable) variables are placed outside of each program. This means that the information contained in these variables can be shared by multiple programs.

In essence, by setting an environment variable on X-Launcher, this will be implemented by the program started.

X-Launcher allows you to define the value of any environment variable, simply by configuring in the [Environment] section of the configuration file:

```
[Environment]  
VARIABLE=value
```

There are also environmental variables of the system, shared by all processes.

To use them, you need to enclose the name of the macro between two % symbols:

1. **%ENVIRONMENT\_VARIABLE%**

The environment variables set in the [Environment] section can be used in all subsequent sections and Key Parameters (section [FileToRun]), and system environmental variables can be used anywhere.

## Auxiliary Variables – [Setup]

The section [Setup] allows you to set some internal variables useful to the compilation of \$ScriptIni\$. The default value of these variables are:

```
[Setup]
AppName=MyApp
AppVer=
UserName=User
Profile=Default
Lang=%LANG%
```

These variables can be referenced in the \$ScriptIni\$, enclosed between two \$ symbols: \$AppName\$, \$AppVer\$, \$UserName\$, \$Profile\$, \$Lang\$.

As for the \$Lang\$, it should be noted that if the system variable \$Lang\$ is not defined, it takes the value “en” (English).

## The FileSystem - [FileSystem]

A very important concept to understand the logic of operation of X-Launcher is the “*FileSystem*”. The FileSystem of X-Launcher is nothing more than a set of internal variables describing a default folder structure

The variables that define the FileSystem are shown in the following table:

| Variable       | Default Value                  | Description  |
|----------------|--------------------------------|--|
| \$ScriptName\$ |                                | Launcher name (ex.: X-Launcher.exe --> \$ScriptName\$= X-Launcher) |
| \$Root\$       | @ScriptDir@                    | Reference for the relative paths                                   |
| \$Temp\$       | @TempDir@\\$ScriptName\$       | Launcher temporary folder  |
| \$Cache\$      | @TempDir@\\$ScriptName\$\Cache | Cache folder that can be used by programs that need it             |
| \$UserName\$   | User                           | User name  |
| \$Home\$       | .\\$UserName\$                 | Folder intended to contain program settings                        |
| \$Bin\$        | .\Bin                          | Folder intended to contain program executable files                |
| \$Lib\$        | .\Lib                          | Folder intended to contain shared libraries from multiple programs |
| \$Doc\$        | .\Documents                    | Folder intended for documents                                      |
| \$Backup\$     | .\Backups                      | Folder intended for backups  |
| \$Download\$   | .\Downloads                    | Folder intended for downloads                                      |

## Syntax Paths

To define routes, we need to use the following syntax:

- \Path Is the relative path to the root folder of the drive \$Root\$
- .\Path Is the relative path to \$Root\$
- ..\Path Is the relative path up one higher level compared to \$Root\$
- ..\..\Path Is the relative path up two higher levels, etc...
- X:\Path Is the absolute path

X-Launcher interprets the relative paths and transforms it into an absolute.

### **Edit FileSystem**

There are two ways to change the filesystem:

- “**User**” mode: settings are written in \$ScriptIni\$
- “**Global**” mode: Settings are written to a file called: **X-Launcher.cfg**, in the same folder as the launcher (@ScriptDir@)

**The settings written to X-Launcher.cfg affect all of the launchers in the folder, while those configured in \$ScriptIni\$ take effect only on the single launcher. The settings in the \$ScriptIni\$ take precedence over those of X-Launcher.cfg.**

The syntax and options for modifying the filesystem are the same for both modes:

```
[FileSystem]
Root=@ScriptDir@
Temp=@TempDir@\$ScriptName$
Cache=@TempDir@\$ScriptName$\Cache
Home=.\$UserName$
Bin=.\Bin
Lib=.\Lib
Doc=.\Documents
Backup=.\Backups
Download=.\Downloads
```

### **Start a Program - [FileToRun]**

The first thing to do to configure the \$ScriptIni\$ is to set the path to the program to be started.

In its simplest form, the \$ScriptIni\$ can be formatted:

```
[Setup]
AppName=MyAppName
[FileToRun]
PathToExe=$Bin$\$AppName$\$AppName$.exe
Parameters=
WorkingDir=
```

The key *AppName*, corresponding to the variable **\$AppName\$**, is the name of the program to start, and its only function is to facilitate the compilation of \$ScriptIni\$.



The key *PathToExe*, corresponding to the variable **\$PathToExe\$**, indicates the full path of the executable to start. Once the path of the executable to start is defined, you have two variables:

- **\$ExeDir\$**: Folder containing the executable
- **\$ExeName\$**: Name (including extension) of the executable

The key *WorkingDir*, corresponding to **\$WorkingDir\$**, specifies the program folder. If not specified, the WorkingDir is **\$ExeDir\$**.

The key *Parameters* allows you to set options from the command line in the program. In this key you can use environment variables defined in the *[Environment]* section.

**Note:** In the key *PathToExe* you can simply define the name of the executable to start, without specifying the complete path. In this case, the executable will be matched, in order, in the following folders:

1. \$WorkingDir\$ (if specified)
2. @ScriptDir@
3. @WindowsDir@
4. @SystemDir@

For example, if you want to run Regedit, just write:

```
[FileToRun]
PathToExe=regedit.exe
```

## Options - *[Options]*

The section *[Options]* allows you to enable or disable the operation of the options in the launcher.

```
[Options]
DeleteTemp=true
MultipleInstances=true
FixAppData=false
RunWait=true
ShowSplash=true
WriteLog=false
```

In this section, the possible values for different keys are Boolean type:

- **true**: enables the option
- **false**: disables the option

### **DeleteTemp**

When enabled, deletes the launcher temporary folder (\$Temp\$) once the program starts (\$PathToExe\$).

**Warning:** This option is only active if it is simultaneously active with the option **RunWait**. Otherwise, there is no way to erase the temporary folder.

### **MultipleInstances**

If disabled, prevents initiating further instances of the program (\$PathToExe\$) while running.

To understand if the program is already running, the launcher can operate in two ways:

1. Check the existence of the process \$ExeName\$ (default)
2. Check the existence of a window (of the program) with a particular title. To activate this mode, you must specify the title to search for using the key “**WinGetProcess**” in the section *[FileToRun]*.

The second mode is particularly suitable in the case of Java applications. In fact, when you start a program written in Java, in reality the process started is “javaw.exe”.

### **FixAppData**

This option allows you to use programs that save their settings to %APPDATA%.

%APPDATA% is a subdirectory of %USERPROFILE%, and its name depends on the language of the operating system. For example:

- OS in Italian, the path is: “%USERPROFILE%\Dati Applicazioni”
- OS in English, the path is: “%USERPROFILE%\Application Data”

Once the path to the directory %USERPROFILE% is set, the option *FixAppData* automatically handles the directory %APPDATA%, renaming the folder depending on the language of the operating system..

**Warning:** This option only works if you set the path to the variable %USERPROFILE%, in the *[Environment]* section.

### **RunWait**

This option allows you to leave the launcher running until the closure of the started program (\$PathToExe\$). This allows:

- a. Delete the temporary directory (\$Temp\$) after the closing
- b. Perform the functions of the section *[RunAfter]*
- c. Perform the RegEdit function in portable mode

To understand if the program is still running, the launcher can operate in two ways:

1. Check the existence of the process \$ExeName\$ (default)
2. Check the existence of a window (of the program) with a particular title. To activate this mode, you must specify the title to search for using the key “**WinGetProcess**” in the section

### *[FileToRun]*

The second mode is particularly suitable in the case of Java applications: in fact, when you start a program written in Java, in reality the process started is “javaw.exe”.

### **ShowSplash**

Enables the splash screen at startup. The properties of the splash screen can be changed in the *[SplashScreen]* section.

### **WriteLog**

Enables the creation of the log file: *@ScriptDir@\\$ScriptName\$.log*

This file is particularly useful to check the settings of the launcher.

# Function References

## The functions of X-Launcher

X-Launcher has several functions, all configurable through \$ScriptIni\$, which can be divided into six groups:

1. Setting environment variables (the equivalent of the DOS command: “set”)
2. Operations of files and folders, run before the program
3. Operations to rewrite files
4. Mixed functions, performed before the initiation of the program
5. Mixed functions, including operations of files and folders, run after the closure of the program
6. Splash Screen: Configure the splash screen displayed at startup

### String Transformation Options

In many aspects, especially in file rewrites, options are available for conversion of strings.

These options are activated by typing the symbol of transformation to be executed after the path, separated by the symbol "|".

The transformation options are:

- = The string is taken as it is, in the case of a relative, it is not interpreted
- ~ Transforms paths into 8.3 format
- %20 Substitutes spaces in paths with the symbol "%20"
- \\ Doubles the backslash "\"
- / Reverses the backslash
- “ It encloses strings

**Example:** \$Root\$ set as the %USERPROFILE%, and create the environment variable %DESKTOP% like this:

```
[FileSystem]
Root=%USERPROFILE%
[Environment]
DESKTOP=.\Desktop
```

The path to the folder %USERPROFILE% is typically:

1. %USERPROFILE%=C:\Documents and Settings\[user]

The variable %DESKTOP% is interpreted as a relative path, and its path will be:

1. %DESKTOP%=C:\Documents and Settings\[user]\Desktop

Using the transformation options, and combinations thereof, the variable %DESKTOP% may be changed as follows:

```
DESKTOP=. \Desktop | \\/%20
```

1. %DESKTOP%=C://Documents%20and%20Settings//[user]//Desktop

```
DESKTOP=. \Desktop | ~
```

1. %DESKTOP%=C:\DOCUME~1[user]\Desktop

```
DESKTOP=. \Desktop | =
```

1. %DESKTOP%=.\Desktop

```
DESKTOP=. \Desktop | " /
```

1. %DESKTOP%="C:/Documents and Settings/[user]/Desktop"

## Setting Environment Variables – *[Environment]*

The section *[Environment]* allows you to set all environment variables required, and is the equivalent of the DOS command “set”.

The syntax is:

```
[Environment]  
VARIABLE=value
```

In this section put active options for transformation of strings.

**The environment variables set in the *[Environment]* can be used in all subsequent sections and Key Parameters (section *[FileToRun]*).**

### Variable %PATH%

The only special case is the variable %PATH%, in which – as in the corresponding DOS command – you can specify multiple pathways, separated by a semicolon (;):

```
PATH=Path1;Path2;Path3
```

Specifying %PATH%, new routes are added to the previous:

```
PATH=%PATH%;@ScriptDir@;.\Path
```

## File and Folder Operations – *[Functions]*

The section *[Functions]* allows you to perform the following operations on files and folders:

- *DirCopy*: Copy folders, subfolders and files
- *DirCreate*: Create empty folders
- *DirMove*: Move a folder, including subfolders and files
- *DirRemove*: Delete a folder
- *FileCopy*: Copy a file or files
- *FileDelete*: Delete one or more files
- *FileMove*: Move a file

### **DirCopy**

Description: Copy folders, subfolders and files.

Syntax:

```
[Functions]
DirCopy=(Source)Path|(Destination)Path|(Option) o
```

The copy is made only if the target folder does not exist. The option “o” (overwrite) allows to overwrite files and then copy even if the target folder exists.

### **DirCreate**

Description: Create empty folders.

Syntax:

```
[Functions]
DirCreate=Path
```

To create multiple folders, you can use the **compact syntax**:

```
DirCreate=Path1\SubPath1;SubPath2|Path2|Path3\SubPath3
```

Which is equivalent to write:

```
DirCreate=Path1\SubPath1
DirCreate=Path1\SubPath2
DirCreate=Path2
DirCreate=Path3\SubPath3
```

### **DirMove**

Description: Move a folder, including subfolders and files.

Syntax:

```
[Functions]
```

```
DirMove=(Source)Path|(Destination)Path|(Option) o
```

The move is made only if the target folder does not exist. The option “o” (overwrite) allows to overwrite files and then make the move even if the target folder exists.

### **DirRemove**

Description: Delete a folder.

Syntax:

```
[Functions]  
DirRemove=Path
```

### **FileCopy**

Description: Copy a file or files.

Syntax: There are two ways depending on whether the copy involves one or more files:

- A Single File: The destination can be a folder, or the full path of the file
- Multiple Files: The destination is a folder

```
[Functions]  
FileCopy=(Source)Path1\file1;file2|(Destination)Path2\|(Option) o  
FileCopy=(Source)Path1\file|(Destination)Path2\ (Or) File|(Option) o
```

The copy is made only if the destination file does not exist. The option “o” (overwrite) allows to overwrite files and copy the files even if the destination exists.

You can use **wildcards**, for example:

```
FileCopy=Path1\*.txt|Path2\
```

### **FileDelete**

Description: Delete one or more files.

Syntax:

```
[Functions]  
FileDelete=Path\file
```

To delete multiple files, you can use **compact syntax**:

```
FileDelete=Path1\file1;file2|Path2\file3
```

Which is equivalent to write:

```
FileDelete=Path1\file1  
FileDelete=Path1\file2  
FileDelete=Path2\file3
```

## **FileMove**

Description: Move a file.

Syntax:

```
[Functions]  
FileMove=(Source)Path\file|(Destination)Path\file|(Option) o
```

The move is made only if the target does not exist. The option “o” (overwrite) allows to overwrite files and make the move even if the destination exists.

## **WriteTo File Operations**

Operations to rewrite files are:

- *StringReplace*: Replacing a string between two others
- *WriteToFile*: Writes any lines specific to a file
- *WriteToIni*: Writes to a file using .ini standards
- *WriteToPref*: Writes to a custom setup file
- *WriteToReg*: Writes to a log file Win98/NT4 (\*.reg)

The file on which the operation is performed must be specified in the section, after the equals symbol (=):

```
[Function=Path\File]
```

## **StringReplace**

Description: Replaces a string between two others in a line.

Syntax:

```
[StringReplace=Path\File]  
Start|End=Substitution
```

Where:

- Start: String preceding the part being replaced
- End: String following the part being replaced
- Substitution: String to be written



You may use the options for string conversion: = ~%20\\

You may specify multiple substitutions in a file.

### **WriteToFile**

Description: Writes lines specific to a file.

Syntax: You need to specify the line on which to write, the corresponding values are the text written in these lines.

```
[WriteToFile=Path\File]
Line1=This is the first line
Line2=This is the second line
...
EOF=This is the last line
```

You may **not** use the string conversion options.

### **WriteToIni**

Descrizione: Writes to a file using the .ini standard.

Syntax:

```
[WriteToIni=Path\File]
Section|Key=Value
```

You may use string conversion options: = ~%20\\

### **WriteToPref**

Description: Writes to a custom setup file.

Syntax: First you need to specify the key *Format*, the format in which to write the settings in the file.

```
[WriteToPref=Path\File]
Format=begin[PREF]mid[VALUE]end
Preferencel=Value1
Preferencd2=Value2
...
```

Where: “begin”, “mid” and “end” should be replaced with appropriate symbols, as shown in the following example.

You may use the string conversion options: = ~%20\\

You may specify multiple substitutions in a file.

**Example**: Mozilla programs save their settings to a file called “prefs.js” (from the profile). By opening this file in text editor, we will find something like this:

```

user_pref("browser.download.lastDir", "D:\\Downloads");
user_pref("browser.download.manager.alertOnEXEOpen", true);
user_pref("browser.offline", false);
user_pref("browser.preferences.privacy.selectedTabIndex", 2);
user_pref("browser.search.selectedEngine", "Google");
user_pref("browser.startup.homepage", "about:mozilla");

```

Osservando la formattazione delle linee, si può facilmente dedurre che il formato delle istruzioni è:

```

user_pref("[PREF]", [VALUE]);

```

Looking at the format of the lines, one can easily deduce that the format of the instructions is:

```

[WriteToPref=%MOZ_PROFILE_PATH%\prefs.js]
Format=user_pref("[PREF]", [VALUE]);
browser.download.lastDir=\\Downloads\\

```

Where: %MOZ\_PROFILE\_PATH% indicates the path to the profile folder, and must be set in the *[Environment]* section.

### **WriteToReg**

**Description:** Writes to a log file Win98/NT4 (\*.reg).

**Syntax:** In the first key, *MainKey*, we must set the main registration key.

```

[WriteToReg=Path\File]
MainKey=PRINCIPLE_KEY
Value1=Content1
Value2=Content2
...
Subkey1|Value=Content
Subkey2|Value=Content
...

```

You may use the string conversion options: = ~%20\\

**Example:** We want to create a log file with the following instructions:

```

REGEDIT4
[HKEY_CURRENT_USER\Software\Audacity\Audacity]
"WantAssociateFiles"=dword:00000000
[HKEY_CURRENT_USER\Software\Audacity\Audacity\Directories]
"TempDir"="C:\\Temp\\X-Audacity"

```

The principle key is obviously the first, enclosed in square brackets, then:

```

[WriteToReg=Audacity.reg]
MainKey=HKEY_CURRENT_USER\Software\Audacity\Audacity
"WantAssociateFiles"=dword:00000000
Directories|"TempDir"=$Temp$\\

```

## Mixed Functions – *[RunBefore]*

The functions of the section *[RunBefore]* are performed before the program *\$PathToExe\$*, and are:

- *RegEdit*: Install registration file \*.reg
- *RunFile*: Start a file or run a command
- *FixDriveLetter*: Rewrite a file changing drive letters with that of Root (*\$Drive\$*)

### *RegEdit*

Description: Install a registration file \*.reg.

Syntax: You can use the compact syntax and specify multiple files in a single command.

```
[RunBefore]
Regedit=Path1\File1.reg;File2.reg|Path2\File3.reg
```

Operations: There are two possible modes of operation:

- Permanent Installation**: The keys contained in the file \*.reg will be installed permanently in the registry. If these keys were already in the registry, they will be overwritten.
- Temporary Installation**: The keys contained in the file \*.reg are installed only for as long as the program is running. If these keys were already present, they will be backed up in the temporary directory, and will be restored at the end of the session. The \*.reg file will be updated with any changes to the keys during program execution.

**Warning**: You may use temporary mode only if the option *RunWait* is enabled. In this case, temporary mode is the default mode, and you can move to the permanent mode using the asterisk (\*):

```
[Options]
RunWait=true
[RunBefore]
Regedit=Path\File1.reg|*
Regedit=Path\File2.reg
; File1.reg installed permanent mode
; File2.reg installed temporary mode
```

### *RunFile*

Description: Start a file or run a command.

Syntax:

```
[RunBefore]
RunFile=Path\File|Parameters
```

**Example**: You can start launching the DOS command interpreter with the appropriate options. For example, the following command displays the text “This is a test” in a DOS window:

```
RunFile=@Comspec@|/k echo "This is a test"
```

### **FixDriveLetter**

**Description:** Rewrite a file changing drive letters with that of Root (\$Drive\$).

**Syntax:**

```
[RunBefore]  
FixDriveLetter=Path\File
```

**Options:** In its default function, only drive letters in CAPS are replaced. You can replace lowercase drive letters using the asterisk (\*). You can also block the replacement of certain drive letters using the “skip=”.

For example:

```
FixDriveLetter=Path\File|*skip=C
```

In this way, the letters that refer to the disk C:\ are not replaced, and since the asterisk is used, even lowercase drive letters will be changed.

**Warning:** In the case of large files that contain many references to local disks, the use of this feature may take a long time for its execution!

### **Mixed Functions – [RunAfter]**

The functions of the section *[RunAfter]* run after the closure of the program \$PathToExe\$, and are therefore only used if the option *RunWait* is enabled.

Features available in this section are:

- *DirCopy*
- *DirMove*
- *DirRemove*
- *FileCopy*
- *FileDelete*
- *FileMove*
- *RunFile*

These functions are the same, as already described, in sections *[Functions]* and *[RunBefore]*.

### **Configuring the Splash Screen – [SplashScreen]**

The section *[SplashScreen]* is used to configure the splash screen displayed at startup. There are three options:

- Image: Sets the image to be used as a splash screen. This image may be of type: **Bitmap**, **GIF**, **JPEG**. If no image is set, the default splash screen is displayed
- Title: Sets the title that appears in the splash screen
- TimeOut: Sets the time the splash screen is displayed (in milliseconds)

**Example:**

```
[SplashScreen]
Image=.\MyImage.jpg
Title=Test
TimeOut=3000
```

# Advanced Concepts

## Change the path of \$ScriptIni\$

The default configuration file, the \$ScriptIni\$ located in the same folder as the launcher, and has the same name but with “.ini” ( @ScriptDir@\ScriptName\$.ini).

There are two ways to use X-Launcher with a different configuration file:

1. Setting the path \$ScriptIni\$ in the environment variable %XCONFIG%
2. Using the command line: --x-launcher-config

### Environment Variable %XCONFIG%

The environment variable %XCONFIG% allows you to set the path to the \$ScriptIni\$ during the opening of X-Launcher.

For example, you may have several configuration files for different applications and use ms-dos batch files to start X-Launcher, with the following simple instructions:

```
@echo off
set XCONFIG=.\Config\Test.ini
start X-Launcher
```

In this example, the “Config”, which is located in @ScriptDir@, contains the configuration file “Test.ini”.

**Warning:** In the variable %XCONFIG% the absolute path of \$ScriptIni\$ needs to be shown. You can use relative paths only if you use a script (batch as in the example) that can interpret the relative paths. In this case, the paths will refer to the script that starts X-Launcher.

### Command Line Option “--x-launcher-config”

The environment variable %XCONFIG%, using the command line option “--x-launcher-config” allows you to set the path to the \$ScriptIni\$ during the opening of X-Launcher.

The syntax of the option is:

```
X-Launcher --x-launcher-config=Path\File.ini
```

The difference between the two ways used to set the path to the \$ScriptIni\$ lies in the fact that, using the command line, **the relative paths are interpreted by X-Launcher, and considered related to the launcher itself.** So, if you run the command:

```
start D:\X-Launcher -x-launcher-config=.\Test.ini
```

the path to the configuration file will be: “D:\Test.ini”.

# Appendix I – Principle Environment Variables

## System Variables

The system environment variables are set by the operating system and you can see the value using the “**set**” command at the command prompt.

You can change the value of the system variables using the section *[Environment]*, and this approach will **only** affect processes initiated by the launcher.

The main variables of interest to us are:

- **USERPROFILE**: Directory containing the user settings (Windows NT systems). Many applications save their settings in this folder, in order to maintain different settings for every PC user.
- **APPDATA**: It is always a subfolder of %USERPROFILE%, and is used by many applications to save different settings for each user. It is also worth noting that, in general, you cannot change its value in the *[Environment]* section, or rather, in doing so, the change has no effect on the portable program. For programs that use %APPDATA% it is therefore necessary to set the directory %USERPROFILE% and enable *FixAppData*.
- **HOME**: Used in Unix operating systems, is the equivalent of the variable %USERPROFILE%. It can be used in many Windows programs derivative of Unix/Linux, and for these programs, has priority over the variable %USERPROFILE%.
- **PATH**: Sets paths search for executable files (see par. 3.2.1). Used in all cases where the portable program needs to know the position of other libraries or programs to work. This is the case with all programs based on GTK libraries, such as Gimp or Gaim.
- **LANG**: Sets language. In some programs, such as Gimp, is the only way to use a different language from that system.

## Variable Details

The following is the list of environment variables known for some specific applications.

| Applicazione | Variabile                    | Descrizione                |
|--------------|------------------------------|----------------------------|
| Amaya        | AMAYA_USER_HOME              | Set Home folder            |
|              | AMAYA_MULTIPLE_INSTANCES=yes | Enables multiple instances |
| Mozilla      | MOZ_PLUGIN_PATH              | Set the plugins path       |
|              | MOZ_NO_REMOTE=1              | Enables multiple instances |
| Gaim         | GAIMHOME                     | Set Home folder            |
|              | GAIMLANG                     | Set language               |
| Gimp         | GIMP2_DIRECTORY              | Set Home folder            |
| GnuPG        | GNUPGHOME                    | Set Home folder            |
| Scite        | SciTE_HOME                   | Set Home folder            |

## Appendix II – AutoIt Macros

The following table contains the main AutoIt macros that can be used with X-Launcher. Further information is available in the manual for AutoIt.

| Macro                      | Description  |
|----------------------------|--|
| <b>@AppDataCommonDir</b>   | Directory %APPDATA% common   |
| <b>@AppDataDir</b>         | Directory %APPDATA% user   |
| <b>@AutoItVersion</b>      | AutoIt version, for example 3.0.81.0   |
| <b>@CommonFilesDir</b>     | Directory %CommonProgramFile%  |
| <b>@ComputerName</b>       | The name of the Computer.  |
| <b>@ComSpec</b>            | The value %comspec%, the command interpreter secondary spec  |
| <b>@CR</b>                 | Carriage return, Chr(13); sometimes used to preserve line  |
| <b>@CRLF</b>               | = @CR & @LF ;occasionally used for line breaks   |
| <b>@DesktopCommonDir</b>   | Common Desktop folder  |
| <b>@DesktopDir</b>         | Current user's Desktop folder  |
| <b>@DocumentsCommonDir</b> | Path to the Documents folder   |
| <b>@FavoritesCommonDir</b> | Favorites common folder  |
| <b>@FavoritesDir</b>       | Current user's Favorites folder  |
| <b>@HomeDrive</b>          | Drive letter containing the home directory of the active user  |
| <b>@HomePath</b>           | Part of the home directory of the active user. To obtain the complete path, use in conjunction with @HomeDrive |
| <b>@HomeShare</b>          | @HomeShare - Name of the server and the share containing the active user's home directory                      |
| <b>@HOUR</b>               | Value of the hour of the clock in 24-hour format. The range is from 00 to 23                                   |
| <b>@LF</b>                 | Line feed, Chr(10); Typically used for line breaks   |
| <b>@LogonDNSDomain</b>     | Logon of DNS Domain  |
| <b>@LogonDomain</b>        | Logon of Domain  |
| <b>@LogonServer</b>        | Logon server   |
| <b>@MDAY</b>               | Current day of the month. The range goes from 01 to 31   |
| <b>@MIN</b>                | Current value of the minutes of the clock. The range goes from 00 to 59  |
| <b>@MON</b>                | Current month. The range goes from 01 to 12  |
| <b>@MyDocumentsDir</b>     | path to the folder My Documents  |
| <b>@ProgramFilesDir</b>    | path to the folder Program Files   |
| <b>@ProgramsCommonDir</b>  | path to the folder Start menu Programs   |
| <b>@ProgramsDir</b>        | path to the folder Programs of the current user (Start menu folder)  |
|                            |  |



|                            |  |
|----------------------------|--|
| <b>@ScriptDir</b>          | Directory containing the running script. (The result does not end with a backslash)                        |
| <b>@ScriptFullPath</b>     | Equivalent of @ScriptDir & "\" & @ScriptName   |
| <b>@ScriptName</b>         | Long Filename running script   |
| <b>@SEC</b>                | Value of the seconds of the clock. The range goes from 00 to 59  |
| <b>@StartMenuCommonDir</b> | Path to the folder of the Start menu   |
| <b>@StartMenuDir</b>       | Path to the Start menu of the current user   |
| <b>@StartupCommonDir</b>   | path to the folder of automatic execution  |
| <b>@StartupDir</b>         | Current user's folder of automatic execution   |
| <b>@SystemDir</b>          | Windows system folder (System or System32)   |
| <b>@TAB</b>                | Tab character, Chr(9)  |
| <b>@TempDir</b>            | Path to the folder "temporary files"   |
| <b>@UserProfileDir</b>     | Path to the folder of the current user's profile   |
| <b>@UserName</b>           | Current user's logon ID.   |
| <b>@WDAY</b>               | Numerical day of the week. The range goes from 1 to 7 and corresponds to the range from Sunday to Saturday |
| <b>@WindowsDir</b>         | Windows folder   |
| <b>@WorkingDir</b>         | Current/Active working directory. (The result does not end with a backslash)                               |
| <b>@YDAY</b>               | Current day of the year. The range is from 1 to 366 (or 365 if it's not a leap year)                       |
| <b>@YEAR</b>               | Current four-digit year  |