Copiaris

User's Manual (rev. 2.3.1)

Table of Contents

Introduction 1

Features 1

Console and graphical (GUI) mode 2

System requirements 2

Limitations in free Lite version 2

User interface languages 3

Migrating from Backup Magic 3

Message from the author 3

Licensing 4

Unlocking Lite version 5

Unlocking Lite version 5

Unlocking with key file 5

Getting started 7

Script 7

Options 7

Inheriting options 8

Creating your first backup script 9

Script editor 11

Running script 12

Progress window 12

Log file 13

File versions 13

Zip compression 15

Shell context menu 15

Tasks 16

Tasks 16

Copy task 16

Process task 17

Message task 18

Wait task 19

E-mail task 19

Registry task 19

Options 20

Script options 20

Script options 20

Advanced script options 20

Log file options 21

Wake-on-LAN options 21

Default options for e-mail 22

Default options for copy tasks 22

Common task options 22

Copy task options 23

Copy task options 23

Copying options 24

Renaming options 25

Filtering options 26

Logging options 27

Versioning options 27

Compression options 28

Copy source options 29

Process task options 30

Wait task options 32

Message task options 32

E-mail task options 33

Set variable task options 33

Set option task options 34

Variables 35

Introduction to variables 35

Option value editor 36

Built-in variables 37

Advanced features 40

Volume shadow copy (VSS) 40

Archive attributes 40

Include & exclude masks 41

Scheduling scripts 43

Optimizing for speed 44

Wake-on-LAN 44

Possible annoyances 45

Advanced scripting 46

Advanced scripting 46

If-then-else conditions 46

User variables 47

Set option task 48

Storing task result in variable 48

Command line parameters 49

Command line parameters 49

Referring options 51

Executable files 51

Exit codes 51

How to 52

Select tasks to run at runtime 52

Prompt for zip encryption password at runtime 52

Pass encryption password by command line 53

How to check if task was executed 53

Sample scripts 54

Find drive by label 54

Download photos from camera **54**

Prompt for directory and zip **55**

Wake sleeping NAS **56**

Code signing with SignTool **56**

Index **a**

1 Introduction

1.1 Features

Copiaris is professional grade high performance file copy and automation tool. It contains native Unicode 32- and 64-bit modules for best performance and simplest administration. Although file copying is main task you can create scripts that do not copy files but do something else, like invoke external command line tools, start/stop services etc.

File copy features:

- Extended length and Unicode file names supported.
- Volume Shadow Services (VSS) (page 40) support, both 32-and 64-bit.
- File versions (page 13) - keep 0 to unlimited number of old versions of files in destination or in another location.
- Zip compression (page 28) with AES encryption and Unicode file names, including option to store each first-level sub-directory in separate zip file.
- Each source directory and copy task can use different copy options and filters if required.
- Merge files from multiple source directories or sub-directories to one directory.
- Batch rename files (page 25) during copying by template, can also rename copied files on file name collision to prevent overwriting.
- Extensive file filtering (page 26) (include/exclude by name or attributes, file size, file age).
- Extensive support for variables (page 35), including nested and user variables. It is possible to prompt for password, directory, file name or text string at runtime using variables. It is possible to pass parameters to variables, for example when reading from registry you can pass default value that is returned when reading registry fails or specified key or value does not exist.

Other features:

- Comes in both console and graphical user interface (GUI) modes (page 2).
- Create advanced scripts (page 46) with if-then-else blocks and user variables.
- Respectable command line support (page 49) - any available option can be passed by command line and program can be controlled completely from command line without using script files.
- Send e-mail (page 19) with attachments and notify about success or failure of remote script by e-mail.
- Built-in Wake-on-LAN (WOL) (page 44) support allows to wake up sleeping remote computers/NAS devices.
- Detailed logging (page 13) and detailed error messages - no solving of cryptic error messages.

Simple to administer:

- Robust and self-contained native Unicode 32- and 64-bit executable - no hassle with runtime requirements.
- Can run directly from network share or flash drive, no local installation is needed.
- Uses plain file copy without creating any backup catalog files to copy or version files. You always know where files are be copied and are stored. No dependencies to restore files, just copy required files back to their original location.

Smart:

- Automatic UAC elevation - prompts for user rights elevation only if current script requires elevated privileges.
- Script launcher automatically starts proper 32- or 64-bit executable according to user operating system.
- Uses minimal disk I/O to offer high performance, especially noticeable when copying over network.
- Shadow copy is only created when there are new or modified files to copy.

- If requesting shadow copy fails because another process is already requesting Copiaris tries again after short wait instead of returning failure.

Quality:

- Unit test suite results consistent quality and reduced possibility that updates introduce regression bugs.

1.2 Console and graphical (GUI) mode

Copiaris comes in two flavors - usual graphical interface (GUI) and console module. Both offer basically the same functionality and can run same script file with same results, but are created for different needs. Most users interact with graphical interface.

Benefits of console mode over GUI

- Console module does not prompt for information nor display any dialog boxes so it can be run unattended and can also be used safely as hidden background process.
- Console module consumes less memory and system resources, especially if you have huge log file with hundreds of thousands of lines. Usual graphical interface displays real-time log and thus keeps entire log session in memory. Console module does not keep log session in memory.

Thus, console module is ideal where robust uninterrupted automation is required with minimal use of system resources.

Some functionality is not available in console mode

Mainly options and variables that ask input from user at runtime by displaying dialog box. Console mode is designed not to display any dialog boxes so that is can run interrupted.

Features that are not supported by console module are marked with `GUI only` in this documentation.

<Console> variable

Variable `<Console>` returns *True* if script is running in console mode and *False* if in GUI mode. You can use it in IF-block condition to run certain tasks only in console or GUI mode.

1.3 System requirements

Since Copiaris is native Unicode application is requires Windows 2000 or newer operating system. There are no other requirements besides some disk space for installation. Copiaris contains 32- and 64-bit modules, so full functionality is available on both platforms.

1.4 Limitations in free Lite version

Free Copiaris Lite version has the following limitations compared to paid full version:

- You can use some tasks only once in your script. Tasks and statements that are not limited in Lite version are set variable, set option, show message and IF/ELSE/ENDIF statements -- these can occur as many times as required in your script. Other task types are only allowed to occur once in your script.
- You can specify only one source directory. Due to this you may also not be able to use <Library> variable if it returns multiple directories.
- File versions can be set to either *disabled (0)* or *unlimited (-1)*, you can't explicitly set number of version to keep, like *5*.

- Values of the following options are disabled (default values will be used): *Copy empty directories*, *Reset archive attributes of copied files*, *Include directories mask*, *Versions directory*, *Zip compress versions*, *Zip comment*, *Zip encryption method*.

1.5 User interface languages

Copiaris comes with multilingual user interface and has built-in translation editor that allows anyone to translate English user interface to their own native language.

To change language of interface or launch built-in *LangEdit* use appropriate commands under *Tools* menu of *Copiaris Editor*.

If you do full interface translation to your own language and would like that translation to be included in future releases of Copiaris you can send us your translation file. See instructions in *LangEdit* on how to make translation.

1.6 Migrating from Backup Magic

If you have backup set file from Backup Magic 1.x then you can open it with Script Editor (page 11) and save as Copiaris script file. However, if you used multiple groups in your backup set file then new Copiaris script that is created from Backup Magic set file may not be optional, although working. During import separate copy task + copy source pair is created for each group that was in backup set. If several groups use the same destination directory then you can move copy sources to one copy task and delete excess copy tasks.

New script architecture and logic of copy tasks is quite different compared to backup set of Backup Magic. It is helpful to read this documentation entirely to understand new features and reasons why something is designed as it is. Although Copiaris is more focused to advanced users and professionals it is still quite easy to set up basic file copy script.

It is recommended to build new scripts from scratch even if you have existing backup set files that can be imported. That way you can lean how to work with new architecture and will find that there are much more options and variables you can use.

Notable changes

While many things are different in Copiaris, starting from architecture of scripts and many new optins and features, there are few changes worth highlighting:

- Improved include/exclude masks provide more functionality and behave somewhat differently compared to Backup Magic. See Include & exclude masks (page 41).
- Copy option (page 24) *Clear archive attribute from source* is enabled by default.

1.7 Message from the author

Dear user,

Thank you for your interest in Copiaris. Hopefully you will find it to be suitable and dependable tool for your backup and automation tasks.

While I have no doubt that Copiaris is high quality software, I still need to apologize about grammar errors in the user interface and in the documentation. There is my strong “Estonian accent” all over!

If you'd like to know more about who made this software please see http://www.moonsoftware.com/about.

1.8 Licensing

Licensing conditions are quite straightforward and user-friendly:

- One copy of the full version may either be used by a single person who uses the software personally on one or more computers, or installed on a single workstation used non-simultaneously by multiple people, but not both. That means for your personal use you may install Copiaris on as many of your computers as required, there is no additional fee for additional computers.
- Family license is economical way to get license for all family members who live on same location for the price of just 2 ordinary user licenses.
- Online activation is not used, which means you can re-install the software as many times as required.
- License costs are very reasonable, especially if you look around and notice how high prices are charged for typical backup software.
- For business users site and worldwide licenses are available.

For pricing and available license types see our web site.

For full License Agreement see license.txt file in installation directory.

2 Unlocking Lite version

2.1 Unlocking Lite version

Free Lite version is functional but has several limitations (page 2). To get full version without limitations you need to buy license at our online store. Immediately after ordering you'll receive your personal license key by e-mail.

Once you have your license key you need to enter it into the program.

1. Start Copiaris Editor
2. Choose **Enter license key** command from **Help menu**. If *Enter license key* menu item is not visible in *Help* menu then you probably have already entered your key and the program already works as full version. To check that you can display About box using *Help* | *About* command and see if *Licensed to* field is displayed.
3. Paste your license key block. Don't try to type it in, it is much easier to just use copy and paste.

Making full version available to all users of computer

In the **Enter license key** window you have option to make the full version available to current computer user only or to all users. All users means all current and future user accounts. Option *All users* is only enabled if you have administrator privileges. Note that on Windows Vista/7/8 even if you have administrator privileges all programs you run are started without administrator privileges by default, which causes situation where *All users* option may be disabled even if you are administrator.

If you are administrator and want to enter key for all users but *All users* option is disabled then close editor without entering key code, then start editor again as follows:

1. Under Windows Start Menu find Copiaris Editor item.
2. Right-click Copiaris Editor item to display context menu.
3. Choose **Run as administrator** command from context menu. Windows asks for confirmation after what editor is started, this time with administrator privileges.
4. Try to enter your key code again, this time *All users* option in *Enter license key* window should be enabled.

Note: If you want to make full version availably only to few users of computer, but not all users, then you need to enter license key for these users only. Log in as desired user, then enter license key as instructed above.

Using key file

It is also possible to store your license key in key file. That is useful if you want to make full version of Copiaris available via network share or want to take it with you on portable drive. See Unlocking with key file (page 5).

2.2 Unlocking with key file

By default your license key that unlocks full version is stored in Windows registry. This is good solution for one computer but if you want to take full version of Copiaris with you on removable drive or access it via network share then key code must be entered in each target computer, otherwise Copiaris starts up as Lite version.

To overcome this you can make license key file that contains your license key block and store it into your Copiaris directory. If you want to share full version through network then you need to put key file to the same directory where Copiaris executable files you are sharing are. On startup Copiaris first checks if key file is present in same directory from where it is started and if so key is read from file instead of registry. That means if key file is present then you don't need to enter key via registry.

Making key file

1. Run Notepad
2. Paste you multi-line license key exactly as it appears in e-mail you received after purchase. It must contain you name followed by more numbers.
3. Save file with name **copiaris.key** to the same directory where your Copiaris program files are. File must be saved with **UTF-8 encoding**.

IMPORTANT: You must select "UTF-8" as value in Encoding list box in Save As window where you specify file name, otherwise international characters in your license key block will become invalid which causes the key file not to be accepted by Copiaris.

Getting key from registry

If you have misplaced your original e-mail with license key you can extract key from registry (if you have full version). There is string value named `Key` under one of the following keys:

HKEY_CURRENT_USER\Software\Moon Software\Copiaris\2.0

HKEY_LOCAL_MACHINE\Software\Moon Software\Copiaris\2.0

HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Moon Software\Copiaris\2.0

Copy value of the Key and paste it to text file as instructed above. Key contains line breaks and once you paste it to editor it becomes multi-line.

3 Getting started

3.1 Script

What is script

With Copiaris you usually create script that contains instructions about what to do when that script is executed. These instructions are called tasks (page 16) and you can use different types of tasks like copy task, start program task, display message task, wait task etc. You can add more than one task to script and these tasks are then executed in given order, one after other. You can also use if-then-else statements (page 46) to run certain tasks only if certain conditions are met.

If you are migrating from Backup Magic (page 3) you can look your script as backup set but it is actually not limited to copying files -- you can use any available task types and copy task is only one of them.

When you create new script in editor then one copy tasks with copy source is automatically created for your convenience. You can delete it if you want to create script that does not use copy task.

Script files

Each script is stored in separate script file. You can create as many script files as required. Each script file is independent file that can be freely copied from one computer to another, like any other document. By using variables (page 35) you can avoid hard-coding file system paths to copy files, so you can create one script that works on many different computers.

You can create script that invokes other scripts using process task (page 17).

Script files have `.copi` extension.

Running without script file

It is also possible to run Copiaris without using script file but passing parameters on command line (page 49). All functionality is available from command line, where you basically build script using command line parameters.

3.2 Options

Task options

In Copiaris you configure tasks by setting their option values. In this documentation I refer these options as task options. Each copy task and source directory can have customized file copy options, if required. For example you can store old versions of files when copying from one source directory but not when copying from another source, or excluding certain types of files when copying from one source directory but not from another.

Script options

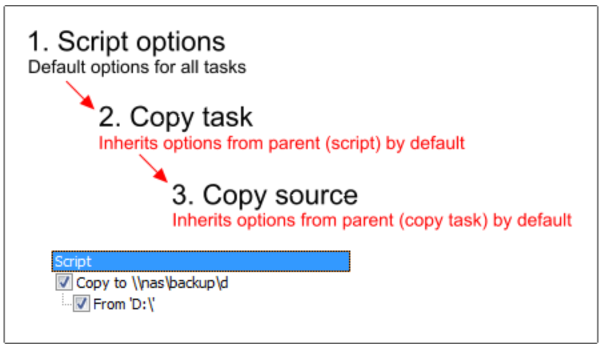
In addition to task options there are also script options. Script options define general behavior of Copiaris during execution of given script and also allow to define default options for all copy tasks used in script file. For example script option *Start minimized* allows you to specify if you script is started in minimized or normal window. Also log file options like *Log file name* and *Log file maximum size* are script options, but logging option *Log copied items* is copy task option, so you can have different *Log copied items* value specified for each copy task, if required.

Inheriting option values

Inheriting value means that option uses value of his parent. By default copy tasks inherit most copy options from script (script is parent for tasks) and copy sources inherit most copy options from copy tasks (copy task is parent for copy sources), making it possible to specify default file copy options for all copy tasks and sources in one place. However, you can override this as discussed in Inheriting options (page 8).

3.3 Inheriting options

Inheriting option value from parent

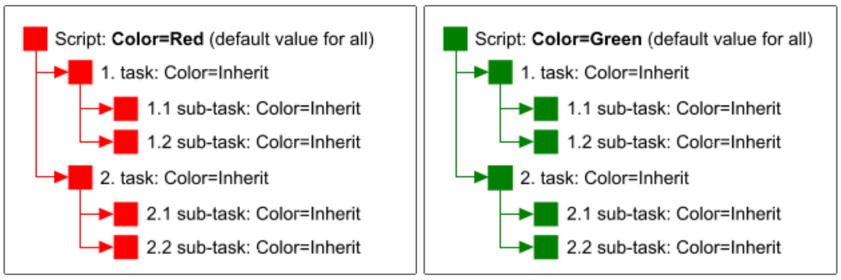

1. Script options
Default options for all tasks
2. Copy task
Inherits options from parent (script) by default
3. Copy source
Inherits options from parent (copy task) by default
Script
Copy to \\nas\backup\d
From 'D:\'

Copiaris allows you to use different options for each task, including each copy task. You can even specify different copy options for each source directory, if required. This is all nice but difficult to manage, especially if you need to assign new option value to multiple tasks simultaneously.

Enter concept of inheriting options. Inheriting value means that option uses value of his parent (if it has parent). So copy tasks can inherit copy options from script (script is parent for tasks) and copy sources can inherit copy options from copy tasks (copy task is parent for copy sources).

By setting value of option to *Inherit* means "use value of parent". If value of parent option changes then value of child option also changes automatically, if child is set to *Inherit* value.

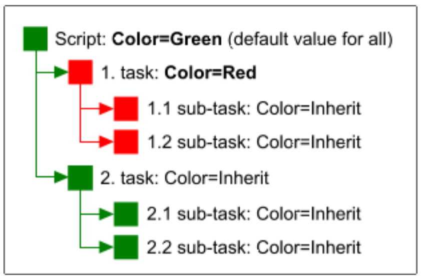
That way you can change script option in one single place and all copy tasks and sources in the script will automatically use that new option value. For example if you want that all copy tasks write names of copied items into log file then instead of setting *Log copied items* to *Yes* for all copy tasks separately you only need to change it in script options and all copy tasks will automatically inherit new value.

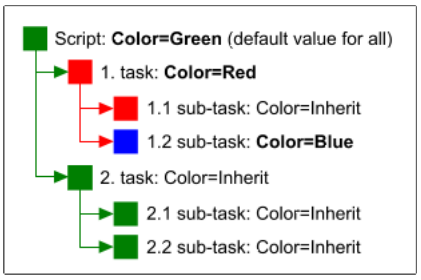

Script: Color=Red (default value for all)
1. task: Color=Inherit
1.1 sub-task: Color=Inherit
1.2 sub-task: Color=Inherit
2. task: Color=Inherit
2.1 sub-task: Color=Inherit
2.2 sub-task: Color=Inherit
Script: Color=Green (default value for all)
1. task: Color=Inherit
1.1 sub-task: Color=Inherit
1.2 sub-task: Color=Inherit
2. task: Color=Inherit
2.1 sub-task: Color=Inherit
2.2 sub-task: Color=Inherit

Graphs above illustrate chain of inheriting values with hypothetical option *Color*. We set value of script option *Color* to either *Red* or *Green* and all tasks and sub-tasks inherit new value for their own option *Color*.

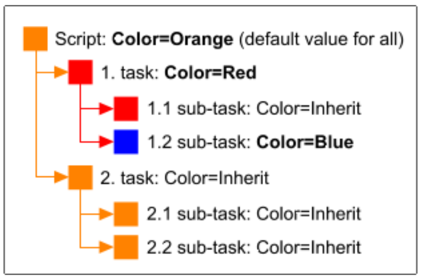
Using custom value instead of inheriting from parent

Using value of parent as default value for most copy task and copy source options is nice but you can also use your own value instead of inheriting from parent by disabling inheriting for specific option(s). That is very powerful concept and allows you to override values of some options, for example use custom value for few copy source options while keeping most other as inheriting from parent/script.


Script: **Color=Green** (default value for all)
1. task: **Color=Red**
1.1 sub-task: Color=Inherit
1.2 sub-task: Color=Inherit
2. task: Color=Inherit
2.1 sub-task: Color=Inherit
2.2 sub-task: Color=Inherit


Script: **Color=Green** (default value for all)
1. task: **Color=Red**
1.1 sub-task: Color=Inherit
1.2 sub-task: **Color=Blue**
2. task: Color=Inherit
2.1 sub-task: Color=Inherit
2.2 sub-task: Color=Inherit

On first image above we have set value of hypothetical option *Color* to *Green* under script options, so it becomes default value for all tasks and sub-tasks. However, for task 1 we have explicitly set *Color* to *Red* instead of *Inherit*. As we can see sub-tasks of task 1 now default *Red* as well, since they inherit *Color* value from parent and value of parent is now *Red*. On second image above we see that you can use custom value where required, at any level, instead of inheriting from parent. Here sub-task 1.2 has *Color* set to *Blue* instead of default *Inherit*, so *Blue* becomes effective value of *Color* for this sub-task.


Script: **Color=Orange** (default value for all)
1. task: **Color=Red**
1.1 sub-task: Color=Inherit
1.2 sub-task: **Color=Blue**
2. task: Color=Inherit
2.1 sub-task: Color=Inherit
2.2 sub-task: Color=Inherit

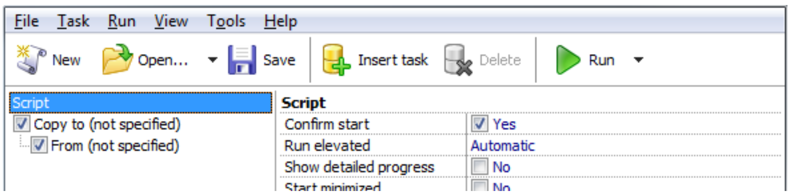
This last image shows that if we now change script option *Color* to *Orange* then all tasks inherit new value automatically except these where you have specified custom value instead of *Inherit*.

3.4 Creating your first backup script

This tutorial helps to create basic script that copies files from source directory to destination directory. Note that it is possible to use Copiaris for other kinds of automation, script does not need to use copy task, it can use and combine any number of tasks.

Create new script

To create new script choose *File | New* command or corresponding toolbar button. Blank script with one copy task and one copy source is created, as seen on image below. Please note that one copy task is automatically added to new script for your convenience, in hope you want to create script that copies files. If you don't want to copy files you can delete the blank copy task from your script and use other tasks.

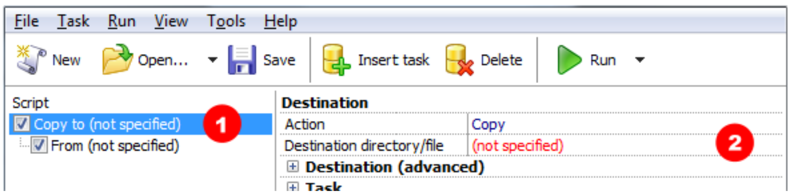

File Task Run View Tools Help
New Open... Save Insert task Delete Run

Script
- ✓ Copy to (not specified)
  - ✓ From (not specified)

Script	
Confirm start	✓ Yes
Run elevated	Automatic
Show detailed progress	☐ No
Start minimized	☐ No

Many options are visible in editor but you actually don't need to change any options, except these few that display text *"not specified"* in red as value. About these I will talk soon. Default values for all options are set so that you can get your script up and running with minimal tweaking.

Set destination directory



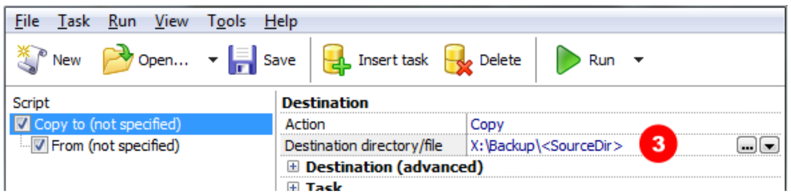
Script
- ☑ Copy to (not specified)
  - ☑ From (not specified)

Destination

Action	Copy
Destination directory/file	(not specified)

⊞ Destination (advanced)
⊞ Task

At right side of editor window options of selected task are displayed, so to specify destination directory click on copy task (1) to display copy task options. Now copy task options are visible at right. Note that main options of each task are visible in topmost, non-collapsed option group. As you can see, main options of copy task are in *Destination* group. You need to specify destination directory -- directory where to copy files to -- by *Destination directory/file* option. This is required value for copy task to work, as you can see from red "not specified" text. Click *Destination directory/file* option with the mouse (2), text "not specified" disappears when option is selected.



Script
- ☑ Copy to (not specified)
  - ☑ From (not specified)

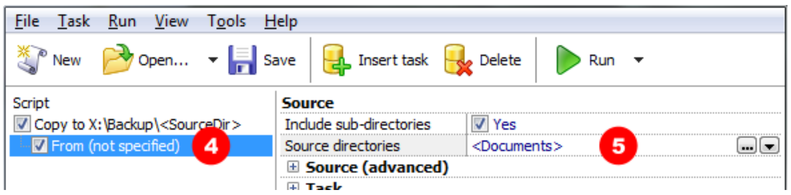
Destination

Action	Copy
Destination directory/file	X:\Backup\<SourceDir>

⊞ Destination (advanced)
⊞ Task

Now enter your destination directory name (3), directory you want to copy files to. You can enter directory by typing, pasting or by value editor that is displayed when pressing edit button ▭. In this example we have used `X:\Backup` as destination directory but you need to specify correct drive letter of your backup drive. Note also that we have included variable `<SourceDir>` as part of destination directory, so destination directory becomes **`X:\Backup\<SourceDir>`** . We don't cover variables in more detail in this tutorial, you can read about them in topic Variables (⧉ page 35), but on this case `<SourceDir>` variable is replaced by name of your source directory at runtime, which is really useful if you want to specify multiple source directories now or in the future. Otherwise files from multiple source directories are copied mixed to the same destination directory, but with his trick we can automatically create subdirectory for each source directory at destination. So my recommendation is to use this variable as visible above.

Set source directory



Script
- ☑ Copy to X:\Backup\<SourceDir>
  - ☑ From (not specified)

Source

Include sub-directories	☑ Yes
Source directories	<Documents>

⊞ Source (advanced)
⊞ Task

Now it is time to specify source directory from where to copy files. Click on copy source (4) to display copy source options at right. Here you need to specify source directory (5). In this tutorial we have specified variable `<Documents>` as source directory. This variable is is replaced by real path of your (My) Documents directory at runtime, thus copying all files and subdirectories from your Documents directory to destination. You can enter any source directory like `D:\Data` here. Note that you can also specify multiple source directories by separating them by semi-colons, like:

```
<Documents>;D:\Data1;D:\Data2
```

Use this only if you want to use same copy options for each source directory. If you want to specify different copy options for each source directory you need to use separate copy source. You can create new copy source using *Task* | *Insert* command. See topic Copy task (⧉ page 16) for more info about copy task and copy sources.

Save script

Now your initial script is ready and your can save file using *File* | *Save* command or corresponding toolbar button. You can save your scripts in your Documents directory and create shortcuts to often used scripts on desktop. After you have saved your script file you can use *File* | *Create shortcut in...* command to make shortcut to it on your desktop for example.

Run script

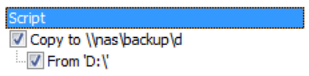
To run your script from editor use *Run* menu or corresponding toolbar button. However, you don't need to open editor to run ready made script file. Instead you can just double-click the script file or shortcut you want to execute. See Running script (⧉ page 12) for more info about different ways to run your scripts.

3.5 Script editor

Script Editor allows to create new and edit existing script files.

Tasks list

Task list at left displays item *Script* plus tasks from current script:



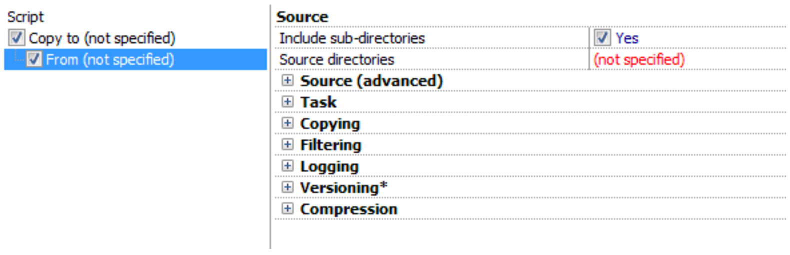
Script
☑ Copy to \\nas\backup\d
☑ From 'D:\'

On image above you see basic script that only contains one copy task that copies files to network location \\nas\backup. The copy task also has one source directory D:\ defined and that is displayed on second level to allow it to have separate options. By clicking any item in tasks list you reveal options of that item in right-hand options list. At top you also see item *Script* that is present in all, even empty, scripts - selecting that allows you to see and modify script options. Script options allow you to change parameters of script itself, like whether to create log file for this script or whether the script starts in normal or minimized window, among others. In addition you can set default options (⧉ page 7) for all copy tasks by selecting *Script* item. By selecting a task you can see or change parameters of that task.

- To see/change options of a task select the task in tasks list.
- To see/change options of script, select *Script* in task list.
- To enable or disable task click check box before task name. Disabled tasks are not executed by default.
- To change order of tasks drag them with the mouse to new location.
- To insert/delete/duplicate task see commands in *Task* menu or in context (right-click) menu of task.
- To move copy source from one copy task to another drag it with the mouse to new location.

Options list

Options list at right displays options of selected task or script.



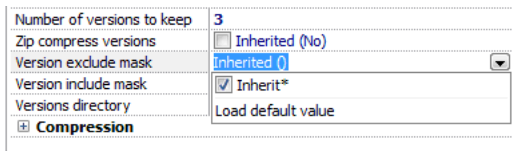
Script
☑ Copy to (not specified)
☑ From (not specified)

Source	
Include sub-directories	☑ Yes
Source directories	(not specified)
⊞ Source (advanced)	
⊞ Task	
⊞ Copying	
⊞ Filtering	
⊞ Logging	
⊞ Versioning*	
⊞ Compression	

In image above we see copy source selected in task list and at right options of selected copy source are displayed. Note that only main options are initially revealed, while other groups are collapsed. Although there are many options available which may look overwhelming at first sight, <u>you actually only need to set few main options to get going and can leave others at factory defaults</u>. Options that require your input are indicated by red *"not specified"*, all other options you can leave as is or customize if required. You don't need to change any options that are in collapsed groups for your tasks to work.

Note also that some option groups may display asterisk * next to group name (like *Versioning** group on image above). That indicates the group contains one or more options with non-default value, or in other words, you have changed options in that group. Makes it easy to see where you have changed factory default options.

Options



Number of versions to keep	3
Zip compress versions	☐ Inherited (No)
Version exclude mask	Inherited ()
Version include mask	☑ Inherit*
Versions directory	Load default value
⊞ **Compression**	

Changing values of options is quite straightforward and for simple integer or string options you just type in new value or where available, can press *Edit* button next to value to invoke special value editor. Some options only allow to use value from pre-defined list of values, on that case available values are listed in drop-down menu. You can invoke drop-down menu by drop-down menu button, by right-clicking or by *Ctrl+Down arrow* from keyboard.

Inheriting option values from parent is very powerful concept and is discussed in more detail in topic Inheriting options (page 8). If an option value is inherited, it displays *"Inherited"* in option list. To specify your own value instead of inheriting value from parent you need to turn off inheriting by displaying drop-down menu and unselect *Inherit* check box. Now you can enter your custom value instead of using value that is inherited from parent.

Option values that you have modified (that have non-default value) are displayed with bold font. You can use popup menu to revert to default value, if required.

3.6 Running script

There are several ways to run a script.

Windows shell

Typically scripts are started by double-clicking on script file or script file shortcut. It is same way you use most documents in Windows environment. You can start a script that way any time you want, for example you can start backup script in the evening after you have finished working on your documents.

If you right-click script file then context menu (page 15) which contains few more Copiaris commands is displayed.

Scheduler

Scheduling scripts allows them to be run automatically at pre-defined times. See scheduling (page 43) for more info.

Script editor

Running script from script editor via *Run* menu or corresponding toolbar button is mostly used when designing a script. You don't need to open editor to run script after you have completed designing your script.

Passing script file as command line parameter

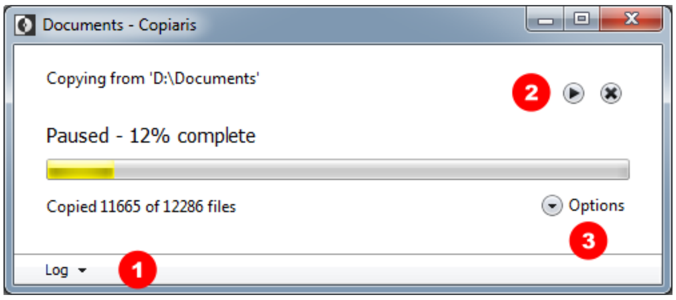
Invoking suitable executable (page 51) by passing script file name as first command line parameter (page 49).

Building script using command line parameters

You can use Copiaris without script file. For this you can build script using command line parameters (page 49).

3.7 Progress window

Progress window is displayed when script is executed. Exception is console mode (page 2) that does not display progress window.



Above you see typical progress of copy operation in less detailed view.

1. **Log button** to show/hide real-time log. Pressing L on keyboard acts the same way.
2. Buttons to **Pause/Resume** and **Terminate** script. You can access these via keyboard by pressing P and T respectively.
3. **Options menu** allows to override some script options for this session only. That means these options are not saved but used only for this session. To change these options permanently see Script options (⧉ page 20). Options menu can be displayed via keyboard by pressing O.

3.8 Log file

By default Copiaris keeps log file for each script file, which contains information about sessions, errors and any other info you have specified by logging options. By default log file is stored in same directory as your script file. Log file can contain multiple sessions and it's size is limited by *Log file maximum size* option, which defaults to about 5 MB (5000 KB). If log file size grows bigger that specified then older entries are automatically deleted.

There are two groups of options associated with logging and log file:

1. Log file options (⧉ page 21) of script. These specify if logging is enabled for given script and parameters like log file name, maximum size etc.
2. Logging options (⧉ page 27) of copy tasks and sources. These task options specify what is logged for given tasks. For example you can specify that given copy task should log names of all copied or excluded files.

Log file is stored in Unicode format (UTF-8). Some older ASCII-only text editors may complain about non-textual file when opening Unicode log files since there are a few binary bytes (preamble) at start of file to indicate Unicode format. If you have issues opening log wiles in third party editors then try plain old Notepad.

3.9 File versions

Copiaris can maintain version history of your files in destination or in special version directory.

What is version history

Version history is keeping old versions of files. Then, in addition to latest version, you have access to older (previous) versions of the file, if required.

Why do I need version history?

Having access to older than latest version of file if often very useful. Imagine you have been working on big document for weeks. Every evening, after finishing days work, you make backup of the document, so you have that precious document file on 2 locations in case something happens to your original file. One morning when you open your document to continue work you discover that in previous days you have accidentally deleted big chuck from your document by mistake -- it is gone! What to do now? You reach for your backup only to discover that it has the same copy of the file since you made backup of already ruined document file, ending up with that same ruined document in 2 places. If you only could go back a few days and get access to previous versions of the document...

Version files

When versions are enabled and file is copied from source to destination and file with that name already exists in destination then existing destination file is not overwritten but is renamed to version file and then new file is copied.

Version file is just your original file with session start date/time added to file name. This is start date and time of session (running script), not last modified time of file. That way all version files generated during one session have the same date/time making it possible to find multiple version files of same age (created in the same session).

For example if your file is named `Letter.doc` and versions are enabled then you may get the following version files:

`Letter.doc`	Latest version
`Letter 120327-114525-CS.doc`	Previous version, created in session started on March 27, 2012 at 11:45:25
`Letter 120120-182321-CS.doc`	Previous version, created in session started on 20, 2012 at 18:23:21

Version directory

Generally you can keep version files in 3 major locations:

1. In sub-directory of destination directory. By default version files are stored in`__Versions` sub-directory of each destination directory.
2. Directly in destination directory -- no sub-directories are created for version files, these are stored in destination directory.
3. In custom directory at different location, even in different computer.

How many versions to keep?

Copiaris can keep from 0 to unlimited number of versions. How many versions is optimal for you depends how often you update your files. If you update several times a day you may benefit from keeping longer history, so you have access to file versions that have a few days old. If you update files once a day you can well keep 3 versions so you can go back up to 3 days. Overall, 3 versions is good default value to have some versions available.

Remember that maintaining versions slows down your file copy somewhat. If versions are enabled, then when copying file Copiaris must check if any versions of file already exist and if there are more than allowed number of versions then also delete excess version files. This all causes more disk I/O, especially noticeable when copying many small files over slow medium like network.

Also, version files take up additional disk space, so when storing large files it may be better to keep only few versions, if any.

Fortunately with Copiaris you can use separate options for each copy source so you can basically assign optimal settings for each source directory.

Compressed version files

It is possible to compress version files into zip files. Compressed version files have similar name to uncompressed versions, but added .zip extension:

```
Letter 120120-182321-CS.doc.zip
```

Searching for version files

Version file names contain token `-CS` after date and time. That is added to make it easier to find all version files from disk, if required. For example you can create script that processes version files. To find only version files you can use mask `"* ??????-??????-CS*"` (without quotes) as value for *Include files mask (page 26)* option of copy source.

3.10 Zip compression

Zip limitations and considerations

Copiaris can create zip files larger than 4GB and add files larger than 4GB to zip file by supporting Zip64 extensions.

You may feel urge to compress contents of your multi-terabyte disk into one zip file but before doing so please consider the following:

- While initial creation of huge zip file is quite straightforward you may see issues when trying to update existing archive with new/modified as update process is time consuming and prone to errors. Often large parts of zip file needs to be re-built just to update one small file. So it is not recommended to use huge zip file as destination when you plan to update it frequently. If you still want to use zip compression consider storing each file in separate zip file.
- Keeping many files in one archive, especially in compressed form, raises risk of losing more than one file if zip file happens to get damaged.
- Potential issues with extra long (extended) file names and paths, longer than 250 chars.

3.11 Shell context menu

Script file is associated with Windows shell so that by double-clicking on a script file Run command is invoked.

Copiaris commands in shell context menu of script file

If you right-click a script file in Windows then context menu is displayed. In addition to common context menu commands it contains the following Copiaris commands:

Run - runs the script. This is default item and also invoked when you double-click a script file.

Edit - opens the file with script editor (↗ page 11).

Run console - runs the script with console module (↗ page 2), instead of graphical interface.

Restoring misbehaving shell commands or file association

If double-clicking script file does not run script or any Copiaris commands are missing from context menu of script file or misbehave you can restore default shell file association by choosing *Make/update shell file association* command from *Tools* menu of script editor. Using that command requires administrator privileges as it restores file association for all users of the computer.

4 Tasks

4.1 Tasks

Task is job or operation, like "copy files from A to B", "run program X" or "wait until drive X becomes accessible".

You make script (page 7) that contains desired tasks in desired order, so they are executed one after other.

See following topics about tasks available in Copiaris. In the future there may be even more different task types available, like e-mail, FTP, registry export etc.

4.2 Copy task

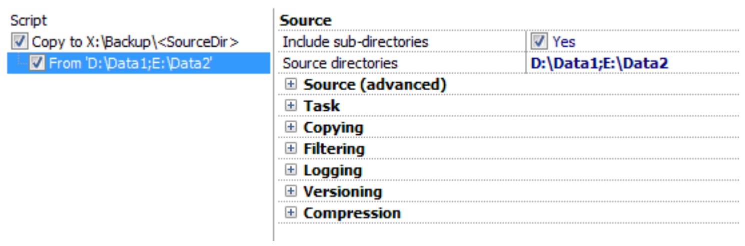
With copy task you can copy files, compress files and also save listing of files to text file.

Copy sources

Copy task itself allows to specify destination directory and copy type, but to specify source directories you need to add one or more copy sources to it. Copy source can be considered sub-task of copy task, but copy task can't function without copy source, because only copy source allows to specify source directories. So each copy task must have at least one copy source.

You can add copy source to your copy task same way you add any task, by *Task* menu. You need to select existing copy task first, then choose *Insert copy source*.

Using one copy source



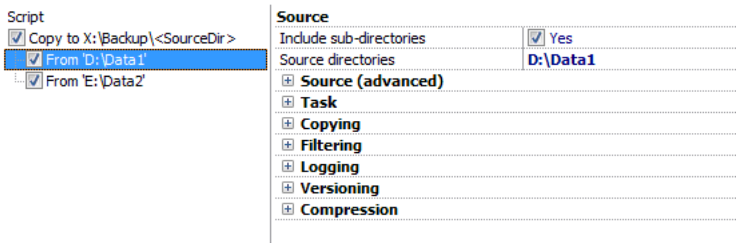
Script
- ✓ Copy to X:\Backup\<SourceDir>
  - ✓ From 'D:\Data1;E:\Data2'

Source	
Include sub-directories	✓ Yes
Source directories	D:\Data1;E:\Data2
⊞ Source (advanced)	
⊞ Task	
⊞ Copying	
⊞ Filtering	
⊞ Logging	
⊞ Versioning	
⊞ Compression	

On image above you see copy task with copy source that has 2 source directories (`D:\Data1` and `E:\Data2`) specified by *Source directories* option. Both these source directories are copied to destination directory `X:\Backup`.

Using multiple copy sources

You may have question about why to use copy source as sub-task and why not just have *Source directories* option as part of copy task itself. It becomes much clearer with the following:



Script
- ✓ Copy to X:\Backup\<SourceDir>
  - ✓ From 'D:\Data1'
  - ✓ From 'E:\Data2'

Source	
Include sub-directories	✓ Yes
Source directories	D:\Data1
⊞ Source (advanced)	
⊞ Task	
⊞ Copying	
⊞ Filtering	
⊞ Logging	
⊞ Versioning	
⊞ Compression	

Script on image above does exactly the same thing as the one on previous image, but this time separate copy sources are used to specify source directories. Each source specifies one directory by *Source directories* option (`D:\Data1` and `E:\Data2` respecively).

Benefits of using multiple copy sources

- Can specify different options for each source directory (copy source), if required.
- Allow user to enable or disable tasks, including copy sources, at runtime when script is executed. See Selecting tasks to run at startup (⧉ page 52).
- Can move copy source to different copy task just by dragging with the mouse.

Strange architecture?

If people first see that copy source (source directory) must be specified "after" copy destination they often find it strange, as in most other programs you typically specify "from" first and then "to". I agree that this indeed looks strange at first. However, the arrangement used in Copiaris has the following benefits:

- Allows to change destination directory or action (copy or compress) in one place by copy task options and all copy sources that belong to this copy task will automatically copy to new location.
- Allows to inherit (⧉ page 8) and override options using logical "default options > copy task > copy source" path.

One early tester wrote: *I really like the "inherit" logic and the destination defined before source logic. The Backup Set "tree" makes it so easy to keep track of what tasks have been defined. Brilliant design!*

4.3 Process task

For process task options see Process task options (⧉ page 30).

Start and stop external program or service

Process task can start/terminate external program (process) or start/stop system service.

By using wait options script can wait until external program returns, then continue executing following tasks.

Note: You usually need administrator privileges to stop service. That means if script is running on Windows Vista or newer platform, it must have been started with elevated privileges, otherwise access to service is denied. See script option *Elevate (⧉ page 20)*.

Host for running external programs

In addition to running simple external programs you can run programs that batch process files and operate on list of files as input. If copy task option (⧉ page 23) *Action* value is set to *Save file listing to text file then instead of copying* files copy task saves file list to text file. It is possible to pass this text file as parameter to external program for further batch processing of these files.

Execute other Copiaris scripts

You can also use process task to run other Copiaris scripts - for example create script that contains 3 process tasks, each one invokes another Copiaris script. Combine that with script option *Select tasks to run at startup (⧉ page 20)* and when running the script you have menu that allows to choose which scripts to run on that session.

4.4 Message task

GUI only

For message task options see Message task options (⧉ page 32).

Message task can display message box with custom message. You can also use it to ask yes/no type of questions from user at runtime and then act according to user response.

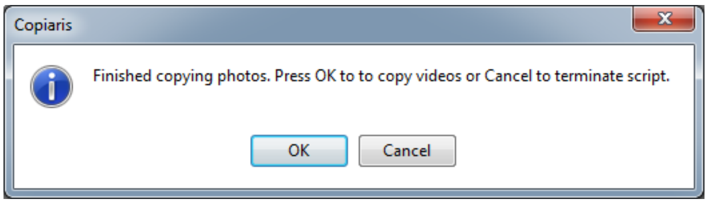
Displaying simple message box

To display simple message box with OK button just add message task and specify message text by message task option *Text*. Note that you can also include variables in your text, these are expanded before displaying message box. For example if you use *"Hello <UserName>! Please connect backup drive and press OK to continue"* text then when message box is displayed variable `<UserName>` is replaced with name of current user.

Allowing Cancel button to terminate script

You can add *Cancel* button to allow user to terminate script. For this specify *Cancel* in *Buttons* in addition to any other button you want to display. That causes processing of script to be terminated if user presses *Cancel* button. For this to work *Fail on* option must contain *Cancel* and *Terminate script on failure* must be selected, both these are set by default.

For example you can use message task to display this kind of message between 2 copy tasks:



Finished copying photos. Press OK to to copy videos or Cancel to terminate script.

This allows user to skip copying videos at that time.

Task result

You can have task result assigned to user variable (⧉ page 48) and process it in script. Instead of typical *Yes* (task success) or *No* (task failure) result message task returns name of button user pressed in message box, so task result that is assigned to result variable (if specified) can be *OK*, *Cancel*, *Yes* or *No* depending which button user pressed. That allows you to create advanced script and call different tasks depending on user response.

Getting name of pressed button

The following pseudo script asks user if he/she wants to encrypt zip file that will be created by script. If answer from user is *Yes* then script assigns password to `ZipPwd` option, causing result zip file to be encrypted. If user answers *No* then password is not assigned and resulting zip file is not encrypted.

```
Show message "Encrypt zip file?" and assign pressed button to variable UserResponse
If <UserResponse>='Yes' then
  Set option ZipPwd=QWERTY
EndIf
Copy task to compress files
```

For the above pseudo script to work you need to assign your variable name (`UserResponse` in this sample) to message task's *Assign result to variable* option, so you can later use IF-block to process user response. Also you must make *Yes* and *No* buttons visible in message by *Buttons* option. Remember that your user variable is set to name of the button that user clicked, it can be *OK*, *Cancel*, *Yes* or *No.*

See Sample script: Prompt for directory and zip (⧉ page 55) for more advanced sample.

Displaying message and terminating script

Sometimes you may want to display message and terminate script immediately after that, like when showing some critical error messages. You can achieve that by enabling *Always* member in *Fail on* task option. Now your message task will always returns failure even if OK button is pressed and since *Terminate script on failure* option is already selected by default, script is terminated.

4.5 Wait task

Using wait task your can insert pause into script or wait until certain drive becomes accessible. Latter is mostly useful if waiting until remote computer/device is ready after waking it up using Wake-on-LAN (↗ page 44).

4.6 E-mail task

With e-mail task you can send e-mail if you have access to SMTP mail server. You can also attach files to e-mail messages and use variables (↗ page 35) in email body, subject or attached file names. You can easily make script that creates zip archive from source files and then sends that file by e-mail to remote location. If you would like to do that it would be good idea to store zip file name in user variable and then use that variable as destination file name and also as attached file.

4.7 Registry task

Registry task allows create or edit registry value. It is also possible to export registry key with sub-keys to .reg file that can be later merged back to registry using system application `regedit.exe` in same or different computer.

If you want to edit or create default value under key then use value @ in *Value name* field.

5 Options

5.1 Script options

5.1.1 Script options

Script options define general behavior of Copiaris during execution of script (⧉ page 7) and also allow to define default options for all copy tasks in this script file.

Confirm start GUI only

Allows to confirm start by displaying message dialog and asking for confirmation. This allows user to prevent running script if it was started by mistake. Useful if script is started by double-clicking script file. You can change message that is displayed to the user by *Confirm start message* under Script options (advanced) (⧉ page 20).

Show detailed progress GUI only

Select if progress window (⧉ page 12) displays detailed or less detailed copy progress. In detailed mode you see more information in progress window, like amount of data copied etc. If you want to temporarily see more details in progress window see *Options* menu in progress window.

Start minimized

Select this option if you want progress window (⧉ page 12) or console to be minimized when script is started.

When finished then

Select action to take after script is finished, like *Close program window* or *Shut down computer*. You can override or set this option in progress window if you decide during running script that you want to take different action.

5.1.2 Advanced script options

<RunCount> ceiling

Variable <RunCount> (⧉ page 37) returns how many times the script has been executed. By specifying ceiling you can make it return value in range of 1..YOUR_CEILING, which means that once ceiling is met, counting start again from 1.

Allow task selection at startup GUI only

When script is started displays window that lists all tasks from script and allows to enable/disable and re-order tasks, making it possible to choose which tasks to run on this session when starting script. See Selecting tasks to run at startup (⧉ page 52).

Confirm start message GUI only

Confirmation message that is displayed to user when script is started and when option *Confirm start* if selected. You can use %s variable in place where you want script file name entered.

Keep console window open

If selected then console window is kept open when program finishes. Applies to console mode only. Note: Console window is always kept open if program has been started by *Run* command in editor.

Run elevated GUI only

Specifies if script is started with elevated (administrator) user rights. With *Automatic* setting elevation is attempted when script contains action(s) that require elevated rights, like creating volume shadow copy (VSS), stopping service etc. Select value *Yes* if you want to force the script to run with elevated privileges, which is useful if the script needs to access resources that are only accessible to elevated user. By selecting value *No* no elevation is attempted even if script contains actions that may require elevation.

Elevation means to elevate user rights on Windows Vista and newer platforms, where by default programs started by administrator are executed in standard user context. To read more about user rights elevation see http://en.wikipedia.org/wiki/User_Account_Control.

Note: Console mode executable typically runs with console privileges and no elevation is attempted unless console mode executable it is started from editor or context menu of script file (page 15) in Windows Explorer.

Show progress on title bar GUI only

Select if you want to see progress percent on title bar of progress window. Useful if script is started in minimized window, then progress is visible on task bar. On Windows 7 and newer platforms progress bar is displayed on taskbar button regardless of this setting.

When finished then notify by e-mail

Specify if and on what conditions notification e-mail is sent when script is finished. You need to specify recipient and additional e-mail options in E-mail section.

5.1.3 Log file options

Log file enabled

Specifies if logging to file is enabled. Log in progress window is always visible regardless of this setting.

Log file name <variables>

Name of log file. If no path is specified then log file is stored in script file directory, unless script file is on read-only media (like read-only network share), which case log file is saved in user %temp% directory.

Log file maximum size

Maximum size of log file in kilobytes (KB). If it is grows larger then older entries are automatically truncated. Specify 0 to allow unlimited log file size.

If you allow logging of file names and copy thousands of files then log file grows very quickly. So don't specify too small size, otherwise even your last session may be truncated.

Log "terminated by user" event as

Specifies what kind of log event is logged when user terminates backup.

5.1.4 Wake-on-LAN options

Enabled

Select to enable wake-up functionality (page 44). You must also specify *Remote IP* and *Remote MAC*.

Remote IP

IP address of remote computer/device you want to wake up, like 192.168.1.10. Required.

Remote MAC

MAC address of remote computer/device you want to wake up. Format of MAC address can be in 001D73A44928 or 00-1D-73-A4-49-28. Required.

Remote port

Port in remote computer where wake-up messages are sent. Typically 7 or 9.

Interval

Specify interval between wake-up messages (in seconds). Wake-up messages must be repeatedly sent, otherwise remote computer/device may go back into standby/sleep mode. Default interval is sufficient on most cases.

5.1.5 Default options for e-mail

Here you can specify default options for sending e-mail, see E-mail task options (page 33). These are used as default options for all e-mail tasks (page 19) and also by *When finished then notify by e-mail* advanced script option (page 20).

5.1.6 Default options for copy tasks

Script options also contain sections to set default options for all copy tasks. Any values set here are inherited (page 7) by copy tasks and copy sources, so they become default options for all copy tasks and copy sources.

5.2 Common task options

All tasks have some common options that define basic task behavior.

Enabled

If selected then task executed, otherwise task is skipped.

Note that there is script option (page 20) *Select tasks to run at startup* which allows to display window to select tasks to run each time when script is executed.

Fail on

Conditions on which task is considered failed. This option is not present for all tasks. If you set option *Failure action* to *Terminate* then here you can choose what exactly causes task to fail. Failure conditions are different for each task type. For example *show message task* has failure condition *Cancel*, which means task is considered failed if Cancel button is pressed in displayed message box.

Name

Name is only used if you want to refer this task from another task or from script. For example you can use set option task (page 48) to change options from script at runtime. Then you assign name to task and then use the same name in set option task to refer your task. Use only upper- and lowercase letters and no spaces and dots in name.

Assign result to variable

If you use advanced scripting (page 46) you may want to known if one of previous tasks failed or not. If you specify user variable name here then that variable is set to either "Yes" or "No" value as soon as task finishes, depending if task was successful or failed. If user variable with given name already exist then value is assigned to existing variable, otherwise new variable is created.

Terminate script on failure

Specifies if script must be terminated if this task fails. Select only if you want to terminate script on task failure, so that following tasks are not executed.

Different tasks can fail on different reasons. For example *Process task* fails if it fails to start external program or start/stop service. Some tasks have additional option *Fail on* that allows to specify on what conditions task is considered failed.

5.3 Copy task options

5.3.1 Copy task options

Destination options

Action

Determines what copy task actually does:

Copy - Copies files from source(s) to destination.

Mirror - Copies files from source(s) to destination plus deletes any files and directories from destination that do not exist in source. <u>Be extra careful when selecting *Mirror* mode, improper use can cause severe data loss. If you mirror from empty directory to root directory of destination drive then all files and directories in destination will be deleted!</u>

Compress all files into one zip - Compresses files from source(s) to destination, adding all source files to single zip file. Name of zip file can be specified by option *Destination directory/file*, but if you don't specify file name by that option, only directory, then default name Backup.zip is used.

Compress each subdirectory into separate zip - Compresses files from source(s) to destination, adding files from each subdirectory of source directory into separate zip file. For example, if your source directory contains 5 sub-directories with files then you get 5 zip files, each named after source sub-directory and containing files from that sub-directory.

Compress each file into separate zip - Compresses files from source(s) to destination directory, each source file in separate zip file.

Save file listing to text file -This special action does not copy anything but saves list of files from source(s) to text file. With it you can use Copiaris' powerful file filtering options to match files and then pass names of these files for processing to external programs via text file. After using copy task to save file names to text file you can use process task (page 17) to launch external program and pass created text file on command line (of course given program must support reading input file list from text file). There are some additional file list options under advanced options section.

Destination directory/file <variables>

Full path of destination directory. If *Action* is set to *Compress all files into one zip* then you can also specify zip file name in addition to directory (if not specified then default name Backup.zip is used). You can use variables (page 35) to generate directory or file name that includes date/time. You need to use full fixed path like *X:\Backup* or UNC path like *\\computer\share*, relative path is not supported.

Advanced destination options

File list encoding

If *Action* is set to *Save file listing to text file* then this value specifies text file encoding. If you want to pass this text file to external program for processing then you need to make sure the external program can read specified encoding.

File list preamble

Text file preamble is kind of file header of few bytes that tells type of encoding used in text file, so program that reads the file knows in what encoding it was saved. Some programs like ImageMagick have problems reading file list if it contains preamble, so it is not enabled by default. Regardless of value of this setting preamble is not stored when encoding is set to ASCII. Used only when *Action* is set to *Save file listing to text file.*

File list quoted

Select to include double quotes around files names in file list that is generated when *Action* is set to *Save file listing to text file*. Required if generated file listing is to be passed to ImageMagick.

Task options

See Common task options (page 22).

Copying options

See Copying options (page 24).

Filtering options

See Filtering options (page 26).

Logging options

See Logging options (page 27).

Versioning options

See Versioning options (page 27).

Compression options

See Compression options (page 28).

5.3.2 Copying options

Allowed file time difference

Allow source and destination file last write time to differ up to specified amount of seconds before they are considered modified. Useful when source and destination volumes use different file systems as some file systems like FAT store file times with low resolution and that causes file times to be slightly different between source and destination.

Copy empty directories

If selected then entire directory structure of source is always created in destination, even directories we do not copy any files to. That means if directory is empty or no files are copied from directory because of exclude filters then empty directory is still created in destination.

Use only when really needed because this feature causes additional disk I/O and makes task slower.

Copy mode

Copy new and modified files - This default copy mode copies new and modified files. How modified files are detected depends on *Use archive attributes* option value. By default last write time of file is used to detect if source file is newer than destination file, but if *Use archive attributes* is selected then archive attributes (page 40) are used instead. If destination is empty then all files are copied, unless you have selected *Use archive attributes* option, on which case only files that have archive attribute set are copied.

Copy all files - All files are always copied, even those that are not modified and also files that already exist in destination. Filtering options are still used. Since there is no need to compare source and destination file times it operates faster, especially when copying many small files over slow medium like network. Can be used to speed up backup if destination is always empty (like when destination directory is created using date variables) or when you need to always force overwriting of destination files.

Destination file attributes to clear

Allows to clear specified attributes of destination files after they have been copied to destination.

Destination file attributes to set

Allows to set specified attributes of destination files after they have been copied to destination.

Overwrite read-only files in destination

Specify if files in destination that have read-only attribute set are overwritten or not.

Reset archive attributes in source

If selected, archive attributes (page 40) of copied files are reset.

Update file times in destination

Select to force file times in destination to be same as in source by specially setting creation and/or last modification time after copying file to destination. Useful if destination file system does not automatically store original file times from source file. Causes more disk I/O, so use only when really needed.

Use archive attributes

If selected uses file archive attribute (page 40) to detect if file is modified, instead of last write time. This can speed up copy process considerably, especially when copying many small files over slow medium like network, but improper use may cause situation where not all files are copied.

Use safe copy

With safe copy file is copied as temporary file first and then renamed. That allows to keep existing destination file intact when something happens during copy process, but requires additional space for temporary storage that may become issue when very large files are copied.

5.3.3 Renaming options

Renaming options in this section allow renaming files in destination during file copy. That means destination file names can be different from source file names, if required. These renaming options work only when *Action* option of copy task is set to either *Copy* or *Compress each file into separate zip.*

Rename by template `<variables>`

Allows renaming of all copied files in destination using same template. For example you can add prefix or suffix to file names, include file modification date or time in file name etc. For example if you use value `<FileName>_<FileDate>.<FileExt>` as template then last modification dates are appended to all destination file names during copying. Use variables to insert dates and other info. It is recommended that when using template to rename files you also select *Rename if exists* option, that allows duplicate files in destination by appending unique serial number to file name in case your template generates same file name to different files.

WARNING! Improper use causes data loss in destination because if given template does not generate unique file name for each copied file then all files are copied with same file name, resulting only one file in destination. Don't assign any value to this field unless you are sure what you are doing.

Rename if duplicate

Contrary to previous option, this option allows automatic renaming of copied files only when files with same name already exist in destination. For example if you are copying file *document.doc* that already exists in destination then instead of overwriting existing file new file is coped as *document (2).doc* (note automatically appended incremental number 2 in file name) and both files are kept. Note also that with this option selected files are never overwritten in destination. Program does not check if file was actually changed from previous copy and just blindly copies again but with new name. This option can be useful when merging files from multiple directories (*Merge sub-directories* source option (page 29)) to empty destination to avoid files with similar names from multiple directories to be overwritten in destination. Also when using *Rename by template* option, in case specified template sometimes generates same name for multiple files.

5.3.4 Filtering options

Exclude directories mask <variables>

Mask(s) of directories to exclude, separated by semi-colon ";". See Include & exclude masks (page 41) for more info about what kind of masks are supported.

Note: Exclude mask is special kind of option that always combines value of current option with inherited value (page 7) from parent at runtime. So any excluded items you specify for copy task do not override excluded items specified by script options, but are added to excluded items of script.

Exclude files mask <variables>

Mask(s) of files to exclude, separated by semi-colon ";". See Include & exclude masks (page 41) for more info about what kind of masks are supported.

Note: Exclude mask is special kind of option that always combines value of current option with inherited value (page 7) from parent at runtime. So any excluded items you specify for copy task do not override excluded items specified by script options, but are added to excluded items of script.

Exclude files with attributes

Files with any of specified attributes are not copied.

Include only files with attributes

If any attribute(s) from this list are specified then only files that have any of specified attributes are copied.

Include symbolic links

Specify if symbolic links and re-parse points are copied or not. Applies to both files and directories.

Include directories mask (copy source only)

If you want to include source sub-directories by mask then write include mask here. Default mask * includes all source sub-directories. Wildcards ? and * supported.

This mask only applies to first level of directories under your source directory. If first level directory matches the mask, all its sub-directories will be included (also these that do not match the mask). In other words, include directories mask is not recursive, but applies to first level of directories only and that is what you usually want.

Note that for copy source (page 16) there are both include directories and exclude directories masks. Typically you want to copy all or most source directories and perhaps exclude some and on that case leave this *Include directories mask* with default value * and specify *Exclude directories mask*.

Include files mask (copy source only)

If you want to include source files by mask then write include mask here. Default mask * includes all files. Wildcards ? and * supported.

Note that for copy source (page 16) there are both include files and exclude files masks. Typically you want to copy all or most files and perhaps exclude some and on that case leave this *Include files mask* with default value * and specify *Exclude directories mask*.

Maximum file age

If specified then only files that are up to specified amount of seconds old are copied.

Minimum file age

If specified then only files that older than specified amount of seconds are copied.

Maximum file size

If specified then only files with size up to specified size are copied.

Minimum file size

If specified then only files with size larger or equal to specified size are copied.

5.3.5 Logging options

Log copied items

Log names of copied files and reason why they were copied (file is new or updated).

If you copy tens of thousands of files and enable this option your log file may grow quite large. You may need to adjust value of *Log file maximum size (page 21)* script option if beginning of your last session is truncated in log file because specified log file maximum size was too small given amount of data logged.

Note also that progress window (page 12) keeps copy of log file in memory for displaying live log, so logging files names when copying vast amount of files also grows memory requirements of the application considerably (though not problem with console module (page 2) as it does not keep log in memory).

Log deleted items

Log names of deleted files and directories.

Log excluded items

Log names of excluded files and directories and reason of exclusion.

Log 'file not found' event as

Specify if you want to log notification, warning or error when source file is not found. See Possible annoyances (page 45) about what can cause "file not found" errors.

Log 'inaccessible source' event as

Specify if you want to log notification, warning or error when source directory is not accessible.

5.3.6 Versioning options

These options specify settings for file versions in case you want to keep multiple old file versions in destination. See Keeping version history in destination (page 13) for more information about file versions.

Each task or copy source can have different options for file versions.

Number of versions to keep

To enable keeping file versions specify up to how many old versions of each file to keep. To disable keeping file versions specify **0**.

Special value **-1** means unlimited old versions. This speeds up maintaining versions because after copying each file to version directory Copiaris does not need to find and delete excess old versions, which causes additional disk I/O, especially noticeable when copying many small files over slow medium like network connection. Drawback is that you may end up with many old versions taking a lot of disk space. To overcome that you can make special script to delete oldest version files now and then.

Zip compress versions

Select to compress version files into .zip files. Each version file is compressed into separate zip file. If you already have version history and enable this option then new versions will be compressed but existing versions are left uncompressed.

Version exclude mask

File mask to exclude certain file types from storing versions. For example you may not benefit from storing multiple versions of audio or video files and then your can add them to exclude mask like `*.mp3;*.mpg;*.mpeg;*.mov`

Version include mask

File mask to version only certain files or file types. By default if you enable versions then you get versions for all files you copy. If you want to version only certain file types, like files made with Word and Excel for example, you can use mask like `*.doc*;*.xls*`

Versions directory `<variables>`

Directory where to keep old versions of files when versions are enabled. This field accepts the following values:

1. Name of subdirectory, without path. By default version files are stored in__`Versions` sub-directory of each destination directory.
2. If left empty or value "." is specified then version files are stored directly in destination directory -- no sub-directories are created for version files.
3. Directory with full path, can be remote directory. If you use full path and your source directory contains sub-directories then you need to use variables (↗ page 35) to construct directory name, otherwise versions from all sub-directories are copied mixed into one directory. The following example instructs Copiaris to use custom version files directory and this directory is used by multiple users for storing version files, so user name is also included in directory name:

```
X:\Old Versions\<UserName>\<SourceDriveLetter>\<SourcePathLessDrive>
```

Note that if versions directory is on different drive compared to destination directory then copy process takes more time since instead of renaming existing destination file to version file in place existing destination file is moved to different disk.

5.3.7 Compression options

Compressed file types

Specify file types that are already compressed to avoid re-compressing these files when zipping. Re-compressing already compressed files wastes a lot of time but gives only minimal gain. Copiaris stores files of these types in zip file without compression.

Important! At time of writing this setting is honored only if copy task *Action* is set to *Compress each file into separate zip* and also for compressing version files.

Zip comment `<variables>`

Enter text to store in zip files created by Copiaris as comment. To disable adding zip comment clear any text.

Zip compression level

Low offers less compression but is fastest and *High* offers best compression but is slowest, while default *Medium* offers balanced performance. To store files in zip without compression choose *Store without compression*.

Zip compression method

Deflate - compatible with old decompression programs.

Deflate64, - also called Enhanced Deflate, newer algorithm and gives slightly better compression with shorter time, but older decompression programs may not support it.

BZip2 - new advanced compression format giving best compression ratio but only some newer decompression programs support it.

Note that program used to decompress these zip files later must support selected method.

Zip encryption method

AES - industry standard encryption with 256-bit key. Decompression supported only by new zip programs.

Classic ZIP - compatible with old decompression programs but not secure by today's standards. Not recommended to use.

Note that program used to decompress these zip files later must support selected method.

Zip encryption password `<variables>`

If you want to encrypt zip files then enter password, otherwise clear all text. The password is stored in script file in plain text format so it is recommended that instead of assigning real password here you use `<UserInput password>` variable to prompt for password at runtime (GUI only) so it is not stored in script file. See Asking zip encryption password at runtime (page 52).

Zip stores full paths

Select if you want to store full paths in zip files. By default relative paths are stored in combined zip and no paths are stored in separate zips. Full path does not store drive letter.

Zip stores Unicode file names

Select to store Unicode file names in zip file header to support international characters in file names in zip file. You need current decompression program that supports Unicode file names to decompress zip files created with that option turned on. Old unzip programs do not support this feature.

Zip use local temporary file for

By default when zipping to removable or network drive local temporary file is used because usually these drive types are slow and it is faster to create zip file locally first and then copy over to destination. If you don't want to create local local zip file first then unselect these drive types.

5.3.8 Copy source options

Source options

Include sub-directories

Choose if sub-directories of source directory are included or not.

Source directories `<variables>`

One or more source directories from where to copy files to destination directory. Variables (page 35) allowed. Separate multiple directories by ";".

Note: If you specify multiple source directories here then same copy options are used for all specified directories. To use different copy options with source directories create separate copy source for each such directory. See Copy task (page 16).

Advanced source options

Delete source files

Specify if source files must be deleted after copying them to destination successfully. BY SELECTING THIS OPTION ALL YOUR SOURCE FILES WILL BE DELETED!

Note: Because Copiaris is smarter than average file copy and backup program it does not delete any files from this source if there were any errors, even one, when copying files. That is to protect you against potential data loss.

Merge sub-directories

Allows to merge files from all source directories, including sub-directories, to one destination directory. No sub-directories are created in destination. Action must be file copy or compress into separate zip files.

WARNING! If there are multiple files with same names in source directories then only one of them remains in destination, unless you use *Rename if duplicate* renaming option (page 25) which automatically renames duplicate files.

Use volume shadow copy (VSS)

Select to enable copying from shadow copy instead of live volume by using Windows Volume Shadow Service (VSS). Copying from shadow copy is slower but is recommended when copying in-use databases and can also be used to copy in-use files on most cases. See Volume shadow copy (VSS) (page 40).

Task options

See Common task options (page 22).

Copying options

See Copying options (page 24).

Filtering options

See Filtering options (page 26).

Logging options

See Logging options (page 27).

Versioning options

See Versioning options (page 27).

Compression options

See Compression options (page 28).

5.4 Process task options

The following are options of process task (page 17).

Process options

Action

Select *Start* to start process or service and *Stop* to terminate process or stop service. If you are terminating process please note that the process is terminated abnormally (killed), any unsaved data will be lost, if there is any. If you are starting a service that depends on other service that is not already running then you need to start that other service first, otherwise starting service will fail.

Process name `<variables>`

If *Process type* is *Service* then name of service you want to start or stop. Service name must be short internal name, not long descriptive name that is displayed in system services list.

If Process type is Executable and *Action* is *Start* then specify full command line to execute, including any command line switches. If executable name or its path contains spaces then you need to put it between double quotes. If *Action* is *Stop* then specify only executable of process you want to terminate.

Note: If you want to run batch file (.bat) or command interpreter command like `mkdir` (make directory), `del` (delete file) etc then you need to to begin your command line with "`cmd /c`" so it is executed by command interpreter.

The following sample command line opens Pictures folder with Windows Explorer. Note that location of pictures folder is given by variable that is replaced by actual pictures directory name at runtime.

```
explorer.exe /n,"<Pictures>"
```

Create directory:

```
cmd /c mkdir "D:\New directory"
```

Start batch file:

```
cmd /c "D:\Batch files\Test.bat"
```

Note: If *Action* is *Stop*, process type is *Executable* and you specify full path then only instance that is running from specified path is ended. Useful if you have multiple instances of given executable running from different locations and you need to stop certain instance. If you don't specify full path then all instances of given executable are stopped.

Process type

Choose process type you are trying to start/stop. Either *Executable* if you are trying to start/terminate program or *Service* if you are trying to start/stop system service.

Advanced process options

Success exit code

Value that indicates successful result of started application, usually 0. If application returned exit code is anything else, task is considered failed. To disable exit code check set this value to -1.

Note: Used only when starting executable and when *Wait for completion* = *yes*.

Wait for completion

If process type is *Executable* and action is *Start* then task waits until executed program finishes running. If process type is *Service* then task waits until service is either completely started or stopped. If you unselect this option and run external program then the program is started but Copiaris does not wait until the program finishes, but continues to execute following tasks immediately after the program has been started. So if your following tasks depend on result of this external program you need to wait until it is completely finished before continuing with following tasks.

Wait timeout

Maximum wait time in seconds. Used only if *Wait for completion* is selected. Default value 0 means wait infinitely. If you know that launched process should return promptly you can set wait time to 30 for example, then if process is not finished in 30 seconds then Copiaris continues executing following tasks. Note that if wait timeout is encountered and launched program is not returned by that time then exit code of the program is not valid (as it is not yet finished running).

Window state

Specify if started program runs in normal, minimized or hidden window. Use hidden window only if you are sure that launched program does not require any user input or does not display any message boxes. Otherwise it may stop running and since program window is hidden you have no way to respond and see what it going on with the program.

Task options

See Common task options (page 22).

5.5 Wait task options

The following are options of wait task (page 19).

Wait options

Drive

Optional. If drive is specified then task waits until specified drive becomes accessible, but not longer than specified by *Timeout*. Drive can be in form `x:`, `x:\dir` or `\\server\share`.

If you use Wake-on-LAN (page 44) functionality then you need to add wait task before copy task and wait until remote computer/device becomes ready by specifying remote drive here.

Timeout

Maximum wait duration in seconds. Specify 0 for infinite wait.

If you specify both *Drive* and *Timeout* and drive has not become accessible after specified timeout then task returns and task result is set to failure.

Task options

See Common task options (page 22).

5.6 Message task options

The following are options of message task (page 18).

Message options

Buttons

Specifies which buttons are visible in message box. When *Cancel* is visible then pressing *Cancel* will terminate script by default, but if you don't like that you can unselect *Terminate script on failure* or remove *Cancel* from *Fail on* options.

Icon

Select which icon is displayed in message box, if any.

Text `<variables>`

Text to display in message box. You can use variables, for example message text *"Disk labeled '<MyDiskLabel>' not found"* contains user variable that has been previously assigned a disk label. Variables are expanded before message is shown, so user will see something like *"Disk labeled 'Backup' not found".*

Task options

See Common task options (page 22).

5.7 E-mail task options

The following are options for e-mail task (page 19):

E-mail options

Priority

Message priority, either *low*, *normal* or *high*.

Subject <variables>

Subject line for your message.

Body <variables>

Message text, can contain multiple lines.

Attachments <variables>

Names of files to attach to message. Separate multiple file names with semi-colon. You can use variables to construct file names. For example by using `<LogFile>` variable you can attach current log file as it resolves to log file name of current script (if logging is enabled). Note that since script is still running, attached log file contains all events up to the moment it was attached and no later events, thus may look partial.

Advanced e-mail options

Advanced e-mail options are by default inherited from script. That way you can configure your mail server name and user account in one location only, under script options, and all e-mail tasks will inherit these settings. Of course you can override any setting in that section for current e-mail task.

Host, Port, Authentication type, User name, Password

Parameters to log in to your mail server. Note that login info like user name and password are stored in script file as plain text and everyone can see them. That means is it very good idea to set up separate user account on your mail server just to send e-mail from script, so that when someone gets to know this login info he does not get access to your e-mail.

Recipient e-mail address

E-mail address of recipient. You can send mail to multiple recipients by separating their addresses by comma. You can also specify name along with address by using the following format: `John Smith <john@johnsmith.net>`

Sender e-mail address

E-mail address of sender. Note that message may not arrive if sender is not specified. If you leave this field empty then Copiaris automatically uses first recipient as sender.

Task options

See Common task options (page 22).

5.8 Set variable task options

The following are options of set variable task.

Set variable

Variable

Name of your variable, required. If user variable with given name already exists then value of existing variable is set, otherwise new variable with that name is defined. When naming user variables be sure you don't use name of existing built variable. You can see list of all built-in variables in variable editor (page 36). Use only alphanumeric characters in name, no spaces and dots. It is a good practice to differentiate your own variables from built-in variables by starting them with prefix "My" for example, like `MyDriveLetter` for example. When defining new variable don't use <> brackets around name, these are only used when resolving variable.

Note that you can only assign values to your own variables, not to any built-in variables.

Value <variables>

Value you want to assign to your variable, if any. Can be empty and can also contain other variables. You can control if variables are expanded before assignment by *Expand variables in value* option.

Expand variables in value

Specify if variables in *Value*, if there is any, are expanded before assigning to variable. By default variables are expanded, so result of variable(s) is assigned to variable.

Operation

Operation allows to select special operation like *Append* (to append new value to existing value, not to replace existing value), *Increment* (to increment existing value by 1 if it is numeric value), *Decrement* (to decrement existing value by 1 if it is numeric value). If *Increment* or *Decrement* is specified then *Value* is not used.

5.9 Set option task options

The following are options of set option task (page 48).

Set option

Task - Specify name of the task whose option you want to set or leave empty if you want to set script option (page 20). You can assign name to task you want to refer by *task option Name (page 22)*.

Option - Name of the option you want to set. That must be short internal name, see Referring options (page 51).

Value <variables>

New value that you want to assign. It can also contain variables. You can control if variables are expanded before assignment by *Expand variables in value* option.

Expand variables in value

If new value contains variables then specify if these variables are expanded before assigning or not.

Task options

See Common task options (page 22).

6 Variables

6.1 Introduction to variables

What are variables

Copiaris has extensive support for variables. Variable is predefined string between brackets, like `<Documents>`, that is expanded at runtime and replaced with value. This `<Documents>` variable expands to (My) Documents directory of current user, which can be `C:\Users\John\Documents` for example. By using variables to reference known file system directories you can make universal script that works on multiple computers and for multiple users. If you specify source directory as `C:\Users\John\Documents` then it only works for user John, but if you use `<Documents>` variable as source directory then it works for any user by returning location of documents directory of that user who is currently running the script.

You can also combine your own text with multiple variables to construct desired value. For example when using `X:\Backup\<UserName>\<WeekOfMonth>` as destination directory it is replaced by `X:\Backup\John\2` at runtime if the script was started by John on second week of the month.

Where you can use variables

Options that support variables have `<variables>`

label displayed in option description. These options usually also support variable editor (page 36) in script editor.

Parameters of variables

Many variables are simple and take no parameters, like `<UserName>` or `<Desktop>`, but Copiaris supports extended variables that allow passing parameters to them. For example variable `<BrowsePath>` allows user to browse directory at script runtime. You can use this variable as source or destination directory to allow user to select source or destination directory at runtime. But `<BrowsePath>` also supports parameter `root`, which allows you to specify root directory for browse dialog, so user can only browse for sub-directory of given root directory. So you can use `<BrowsePath root="D:\Data">` to specify `D:\Data` as root directory for browse dialog and on that case in displayed browse dialog user can only select sub-directories of `D:\Data`.

Nested variables

Variables can be nested, which means output of one variables becomes input of another.

As example of nesting variables lets use same `<BrowsePath>` variable as in previous topic, but this time we use another variable to specify `root` parameter. By using `<BrowsePath root="<Documents>">` we get `root` parameter value by resolving `<Documents>` variable, which effectively means user can only browse for sub-directory of his `(My) Documents` directory.

If any nested variables are present then innermost variable is always resolved first. In this case `<Documents>` is resolved first and then `<BrowsePath>`, making it possible to use resolved value of `<Documents>` as input for `<BrowsePath>`.

User variables

Special kind of variables is user variables which are defined by script creator. User variables are discussed in dedicated topic (page 47).

System environment variables

You can also use system environment variables like %TEMP%. If Copiaris has built in variable that returns same value as system variable (%TEMP% and <Temp> for example) then it is better to use built-in variable because directory values returned by built-in variables never end with backslash, which may not be true about system variables. Also resolving built-in variables is quicker.

System variables are resolved after resolving all built-in variables so system variables cannot be nested inside built-in variables.

When variables are expanded

Variables are generally expanded when encountered. Destination directory is resolved when copy task is executed but after source directories are resolved, because you can use source directory path/name for constructing destination directory name using `<Source...>` variables.

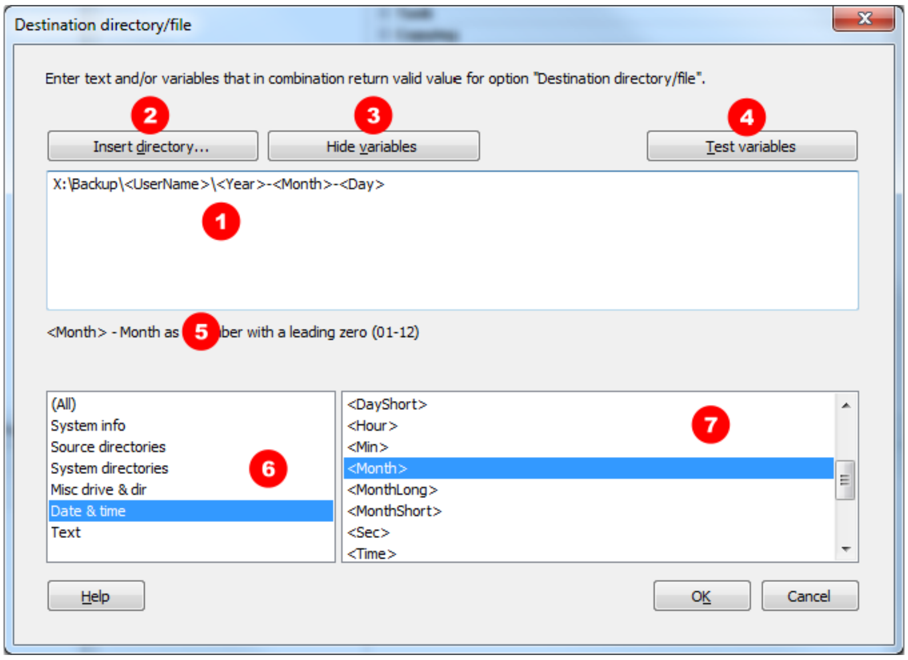
Resolving variables when they are encountered usually poses no problem but if you need to ask user input by variable then it may be helpful to do that when script is started by using user variables (⧉ page 47).

6.2 Option value editor

Option value editor is invoked by script editor to specify value for string options that allow variables (⧉ page 35) to be used in value. It is much easier to construct option value using this window that by typing into edit box of option value.

Contents and layout of this window can vary and controls you see depend on option you are editing.

In following image you see option value editor invoked to assign value to *Destination directory.*



X:\Backup\<UserName>\<Year>-<Month>-<Day>

<Month> - Month as ber with a leading zero (01-12)

Window elements:

1. **Value** of option you are editing. Whatever you enter here must be valid value for option whose value you are editing. You can enter variable at caret position by double-clicking variable name in variables list (7), or you can also type. If you are editing value of *Exclude files mask* or *Exclude directories mask* then you see 2 edit boxes here. Read-only edit box at left displays inherited exclude mask for your convenience as these 2 options always add their value to value inherited from parent.
2. **Insert directory** or **Insert file** button allows to insert file system directory or file name. This button may not be visible if editing an option that has nothing to do with files or directories.
3. **Show variables/Hide variables** button to expand/collapse window to show/hide variables list at bottom of window. By default variables list is not visible.

4. **Test variables** button. If you use any variables in value (1) then you can use this button to see resolved value. If you use variable that takes parameters then you can test to see if you have passed parameters properly. Note that you can't test user variables -- if user variables are used in value then you'll see "unknown variable" error.
5. **Syntax of selected variable**.
6. **Groups of built-in variables**. If *(All)* is selected then all built-in variables are displayed in variables list (7), else only variables of selected group are displayed.
7. **Variables list** displays all built-in variables (page 37). Variables that are grayed cannot be used for current option. Double-click variable name to insert it into value at caret position. At time of this writing user variables (page 47) are not listed here but can also be used.

6.3 Built-in variables

Full list of all built-in variables is visible in option value editor (page 36) along with short descriptions. While most variables are simple and do not require further explanation here are listed only these variables which may need additional instructions.

<Admin> - Returns *yes* or *no* string, depending if user account under which script was executed has administrator privileges or not. Can be used in IF-block condition to execute some tasks only when user has or has not administrator privileges.

<BrowseFile [title=][dir=][filename=][ext=][owprompt]> GUI only

Returns file name by displaying common dialog like typical open/save dialog to allow user to select existing file name or enter new file name. You can use other variables to provide value to parameters, like in example below we use variable `<Desktop>` to set value of `dir` parameter.

Parameters (all optional):

`title` - Title displayed in common dialog

`dir` - Initial directory

`filename` - Default file name

`ext` - Default file extension

`owprompt` - Prompt user to overwrite file if file with given name already exists

Example:

```
<BrowseFile title="Select destination zip file" ext="zip">
```

```
<BrowseFile title="Select destination zip file" filename="Backup of <DateShort>" ext="zip"
dir="<Desktop>" owprompt>
```

<BrowsePath [prompt=][root=]> GUI only

Returns directory name by displaying *Browse for folder* common dialog to allow user to select an existing directory.

Parameters (all optional)

`prompt` - Text displayed in window

`root` - Root directory. If specified then user can only return that directory or any subdirectories of it, but can't navigate to any other parent directory.

Example:

```
<BrowsePath prompt="Select source directory:">
```

`<BrowsePath prompt="Select source directory:" root="<Documents>">` - allow browsing only (My) Documents directory and sub-directories of it.

<Console> - Returns *yes* or *no* string, depending if script is running in console or GUI module (page 2). Can be used in IF-block condition to make one script that does different things depending if it has been running in console or in GUI.

<Drive [label=][serial=][file=][dir=][type=]> - Returns drive by specified parameters in form *X:*. If specified drive is not found then task where this variable is used likely fails. Typical use is finding drive letter of removable drive/flash disk/card reader by label to overcome issue where drive letter may change over time, depending on what other drives you have currently connected.

Parameters (one required):

`label` - Finds drive letter by specified label, case does no matter.

`serial` - Returns drive letter by specified volume serial number. It must be in 8-char hex format, case does not matter.

`file` - Returns drive letter where specified file exists. If file is in root directory specify just file name like *File.ext* or is in subdirectory then specify *Dir\File.ext*

`dir` - Returns drive letter where specified directory/sub-directory exists. Specify directory like *MyDir* or *MyDir\SubDir*

`type` - Returns first drive of specified type. You must specify one of the following value: *removable*, *fixed*, *network* (returns only mapped network drive), *dvd* (returns any optical drive like CD-ROM etc) or *ram* (returns RAM/memory drive).

Examples:

`<drive label="MyCameraPhotos">` finds drive letter for of your connected memory card labeled *MyCameraPhotos*.

`<drive file="Copiaris\Copiaris.exe">` finds drive letter of drive where there is directory *Copiaris* that contains file *Copiaris.exe*. You can use this to find drive letter from where you run Copiaris for example, although there is dedicated variable `<AppDrive>` for that.

`<drive type=dvd>` returns drive letter of first optical drive (DVD-ROM, CD-ROM etc).

<ErrorCount>, **<WarningCount>**- Returns number of logged errors or warnings.

<ErrorLevel> - Returns current error level that can be *0* (no errors or warnings logged), *1* (warnings logged) or *2* (errors logged). It is useful in IF-condition to execute certain tasks only if there are errors or warnings logged. For example you can add IF-block to the end of your script with condition `<ErrorLevel> > 0` which causes the IF-block to be executed only if there are errors or warnings logged. In that IF-block you can execute task that notifies you about script failure, for example sends log file by e-mail.

<Library [name=][exclude=]> - Returns semicolon separated list of directories that belong to library specified by name, or if name is not specified then list of all directories from all libraries is returned (entire library). You likely use this variable as source directory to backup files from your library. Note that it often returns multiple directories separated by semicolon as one library may contain multiple directories from different locations.

Note: Libraries are supported on Windows 7 or newer only. On older platforms always returns (My) Documents directory.

Note: If this variable returns more than one directory you can't use it in free Lite version as source directory since Lite version can only operate with one source directory. Upgrade to full version (page 5) to eliminate this limitation.

Parameters (specify either `name` or `exclude`, not both):

`name` - Names of libraries you want to get directories of. To specify multiple names separate names by | like *"Pictures"|"Videos"*. If you omit `name` parameter then directories of all libraries will be returned, excluding only those whose are specified by `exclude` parameter (if present).

`exclude` - Specify names of libraries to exclude. Useful if you don't specify `name` parameter but don't want to return directories of all libraries.

Examples:

`<library>` - Returns all directories of entire library.

`<library exclude="Photos"|"Videos">` - Returns all directories of entire library, except those from Photos and Videos libraries.

`<library name="Photos"|"Videos">` - Returns only directories of Photos and Videos libraries.

<Registry key= value= [default=]> - Return value from registry.

Parameters (`key` and `value` required):

`key` - Name of registry key in form *HKEY_CURRENT_USER\Key\Subkey.* Note that Copiaris needs administrator permissions to read from most other root keys than *HKEY_CURRENT_USER*.

`value` - Name of value to read.

`default` - Default value to return if value in registry is empty or there was error reading value from registry. This parameter is optional.

<RunCount> - If you have specified *<RunCount > ceiling (⧉ page 20)* under script options (⧉ page 20) then this variable returns value in range 1..YOUR_CEILING, which means that once ceiling is met, counting starts again from 1. You can use it to create custom backup rotation instead of using `<DayOfWeek>` and similar variables in destination directory name.

<UserInput [prompt=][default=][password][confirm]> GUI only

Returns text that user enters. Can be also used to ask zip encryption password from user at runtime, so it is not stored in script file in plain text.

Parameters (all optional):

`prompt` - Text that is displayed to user along with text entry field.

`default` - Default text that is already entered in text entry field when prompt is displayed. User can change it or leave it as is.

`password` - By specifying `password` parameter input window is displayed to allow user to enter password. If this parameter is used then value that user returns is not logged, so be sure to specify it when asking for password, otherwise user password may end up in log file in plain text as typically result of variable expansion is logged.

`confirm` - Use along with `password` parameter to force checking of entered password. If present then input window contains 2 fields, "password" and "confirm password". Use confirming when you assign password to encrypting something, in that case it is important that user does not make typo when entering password, otherwise it is not possible to decrypt later. If you ask password for decryption you usually don't want to display confirmation.

Example:

`<UserInput prompt="Zip password:" password confirm>` - Asks zip encryption password from user at runtime. Can be used as value of *Zip encryption password* option to ask password at runtime instead of storing it in script file in plain text. See also Sample script: Prompt for directory and zip (⧉ page 55).

7 Advanced features

7.1 Volume shadow copy (VSS)

Windows contains technology called Volume Shadow Copy Service (VSS) which allows to create shadow copy (snapshot) of volume and copy files from snapshot instead of live volume. Snapshots have two primary purposes: they allow the creation of consistent backups of a volume, ensuring that the contents cannot change while the backup is being made; and they avoid problems with file locking, so files currently in-use/open by programs can be copied.

Note: VSS is not available on Windows 2000

Benefits of shadow copy

- Possible to copy files that are otherwise in-use by programs.
- Possible to get consistent copy of set of files, like database files that are in use by database server. Major database servers are VSS-aware, so they get notified when shadow copy is requested by backup application like Copiaris and that allows them to save all database files to disk in "good" state before shadow copy is returned.

Drawbacks of shadow copy

- Creating shadow copy can take several minutes or more.
- Applications can't request shadow copy while one shadow copy is already being created for another application (or another instance of the same application).
- Complicated COM architecture with many components, often from multiple manufactures. May be difficult to find reason of failure if something VSS-related does not work.
- Administrator privileges/elevation is required to create shadow copy.
- On 64-bit Windows Vista or newer platform VSS-client must be also 64-bit application.

Using VSS in Copiaris

If you want to use VSS in Copiaris you need to enable it for copy source (↗ page 16) by *Use volume shadow copy (VSS) (↗ page 29)* option. That way you can only enable it for select sources if you have more than one source.

See Possible annoyances (↗ page 45) for details about potential "file not found" errors when using VSS with Copiaris.

7.2 Archive attributes

Archive attribute is property of each file that tells if that file has been modified since last time when that property was reset. It is typically used by backup applications and is reset when file is backed up. When file is modified then archive attribute is set so next time backup application runs it can detect that modification.

Ways to detect if file is modified

Compare last write time of source and destination file

This is the default method.

Benefits

- Robust, always works even if you use different copy/backup applications simultaneously

- Maintenance free

Shortcomings

- Slow over network. Needs to query destination for last write time of each matching file, even those that are not modified, which can be slow over network connection. Especially noticeable when source directories contain many files.

Recommended when/for

- Copying from local disk to local disk

OR

- Robustness is needed and file scan time is no issue
- Casual users

Archive attributes

Benefits

- Very fast because uses local archive attributes to detect files that need to be copied. No need to query destination.

Shortcomings

- Very easy to miss files to copy because needs special considerations when setting up. If another program clears archive attributes or some source files have archive attribute not set when setting up backup then these files are not copied. Also, when you change destination drive or directory only files that are modified since last backup are copied.
- Needs some "maintenance" to avoid previous issues

Recommended when/for

- Many files need to be scanned and either source or destination is remote computer
- File scan must be as quick as possible
- Advanced users and professionals

Side effects & dangers

Using archive attributes can give very quick file scanning times but since it does not communicate with destination to see what files really exist there it is very easy to end up with incomplete backup. That is why it is only recommended to advanced users who are aware of the shortcomings.

Main issue with archive attributes is that if you change destination drive or directory then files that you have copied in the past to previous location are not copied again until you either set archive attribute for all source files manually or force copying of all files using *Copy all files* as *Copy mode*.

Special uses of archive attributes

Copying only new & modified files to empty destination directory

Sometimes you may want destination directory to contain only new and modified files since last backup and then you are forced to use archive attributes, because otherwise all files are copied since they do not exist in destination.

7.3 Include & exclude masks

Include & exclude masks that are part of copy task's filtering options (page 26) allow you to use typical file mask wildcards ***** and **?** but also extended masks that contain some or full path to refer certain directory, root directory etc.

Evaluation order

Copiaris evaluates include mask first and then exclude mask. If you specify * as exclude mask then obviously nothing is copied.

Testing masks

To see if your exclude masks work properly you can enable logging of excluded items under Logging options (page 27). Then all excluded items are logged, including detailed reason of exclusion.

Supported mask types

Basic masks that match any file in any directory or any directory

Basic masks do not contain any path information, so they apply to all files.

```
file??.*
```

or

```
*.ext
```

In exclude directories mask, to exclude any directory, including subdirectories, named "History":

```
History
```

Extended mask containing some path info

In exclude files mask the following mask excludes all *.tmp files that are in any directory named History:

```
*\History\*.tmp
```

Extended mask with full path

Mask with full path and no wildcards only exclude one certain file in certain directory. You can use that kind of mask when you wand to skip one file because it is in-use by another program for example and you don't need to copy it:

```
X:\Database\Database.lock
```

Adding wildcard to full path can exclude matching files in certain directory only. Following example excludes all *.ext files that are in X:\Test directory, but *.ext files from other directories are not excluded:

```
X:\Test\*.ext
```

In exclude directories mask, to exclude certain directory including its sub-directories, write its full path:

```
X:\Temp
```

Extended mask with root directory indicator

If mask starts with "\" (and not "\\") that means following file or directory name is matched only if it is in drive root. It is useful if you want to refer root directories on multiple drives to avoid specifying drive letter. If you want to refer root directory of only one drive then you can include full path like discussed in previous section.

Following file mask excludes all files that are in drive root but includes files in sub-directories. Useful if source directory is drive root, like "x:\":

```
\*
```

You can exclude files from certain root directory. Following exclude files mask will exclude all files that are in directory Data that is in drive root, but if there are any other Data directories that are not in drive root, then files from these are not excluded:

```
\Data\*
```

To get similar result you can use exclude directory mask. This is also faster because files from Data directory does not need to retrieved and checked against file masks:

```
\Data
```

Following extended exclude files mask excludes only *.doc files that are in sub-directories of Documents directory that is in drive root. *.doc files from other Documents directories, if any, are not excluded. Also are not excluded *.doc files from Documents directory itself that are not in sub-directories of it.

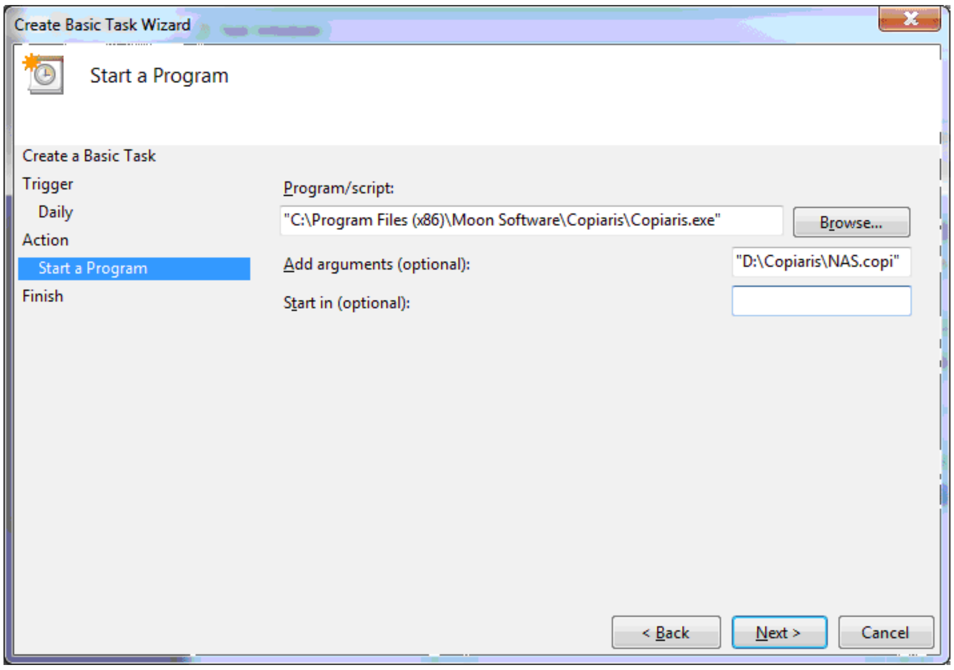
```
\Documents\*\*.doc
```

7.4 Scheduling scripts

To schedule script to run at specified time or event you can use Windows Scheduler.

Windows 7

1. Open **Task Scheduler** (usually under Start Menu | Accessories | System Tools).
2. Choose command **Create Basic Task** (from **Action** menu or from right panel). Now wizard is displayed.
3. First name your task, name is used to differentiate it from other existing tasks. You can use name similar to your script name so you know what this task does by just looking at this name.
4. Next step is to specify **trigger**. Trigger is event that triggers execution of task, it can be certain time/date event or an operating system event. This basic task creation wizard lists most common triggers only, it is possible to specify advanced triggers later via advanced properties of scheduled task.
5. After your trigger is set up, next step is to specify **action**. Specify **Start a program** action here and press Next
6. Following **Start a program** page is displayed:



Program/script: "C:\Program Files (x86)\Moon Software\Copiaris\Copiaris.exe"

Add arguments (optional): "D:\Copiaris\NAS.copi"

Start in (optional):

You need to specify the following:

Program/script - Full path to Copiaris executable in your system, between double quotes. You can use *Browse* button to browse right executable. In Copiaris installation directory there are several executable files, but you need to choose `Copiaris.exe` if you want to run graphical interface that displays progress and allows to pause/cancel script or `CopiarisCon.exe/CopiarisCon64.exe` if you need to run console mode executable. Console executable must be always used to run your script on background without interaction or to run your script under different user account (hidden). see topic Console and graphical (GUI) mode (page 2) for more info about different modes.

Add arguments - Here is where you specify what script to run. Script name must also contain full path if it is not in same directory as executable and must be between double quotes. To paste script file name here you can copy it from Copiaris Editor - open your desired script file in editor, then choose *File* | *Copy file name*. This parameter is required, otherwise Copiaris executable does not know what script to run.

Note: To run script in minimized window add parameter `Minimize=Y` after your script file name in *Add arguments* field (separated by space). You can use any other command line parameters (page 49) as well.

Now you can finish creating your scheduled task by pressing Next. New task is listed in *Task Scheduler* window (if you can't find it be sure *Task Scheduler Library* is selected on left panel). To **test your new task** right-click it and choose *Run*.

7.5 Optimizing for speed

If you need to copy files as fast as possible then here is list options that can influence speed of finding and copying files. Value that results fastest speed is specified.

- **Use archive attributes = yes** (default = false). By using archive attributes you can speed up finding modified files considerably if you are scanning many files over slow network connection for example. However, before using archive attributes be sure to read about side effects (page 40).
- File versions (page 13) can be slow, the more versions you keep the slower. If you want to use versions but require as fast as possible performance then use special value **Number of versions to keep = -1** (default = 0), which enables unlimited versions and is fastest, but you need to delete excess old versions by yourself.
- **Reset file attributes of copied files = false** (default = yes). You can disable this if you don't use archive attributes.
- **Use safe copy = false** (default = yes). Safe copy means to copy into temp file first and then rename, which is safer but takes additional rename operation that can slow down especially if copying many small files over slow network.
- **Copy empty directories = false** (default = false). If you don't need to copy empty directories then be sure that option is not selected.

7.6 Wake-on-LAN

Wake-on-LAN (WOL) is a computer networking standard that allows a computer to be turned on / woken up remotely, by a network message (magic packet).

If you want to copy files from/to computer or network attached storage (NAS) device that may be in standby/sleep state you can configure your script to send wake-up message to that computer/device, wait until the computer/device wakes up and is ready and then start copying to/from the computer.

Note that the computer/device you want to wake up must be configured to wake up on WOL messages. Some devices may be WOL-enabled by default while others may require adjustment. Computers often have WOL-related options in BIOS setup and network interface card drivers also may have their own WOL-related options.

Copiaris sends wake-up messages while script is in progress and stops sending wake-up messages when script is finished. Remote computer/device most likely goes back into sleep/standby mode if it stops receiving wake-up messages. If and how quickly that happens depends on specific device configuration.

Waiting until remote drive becomes accessible

After remote computer/device receives wake-up message it takes time until it gets ready and fully accessible. It may take are as long as 2-3 minutes for NAS device to start. If you need to copy files to/from remote computer after waking it up then script must contain wait task that waits until desired drive becomes accessible before copy task.

Imagine you have NAS device named NAS and it contains file share named `Backup` that you usually access by path `\\NAS\Backup`. You have it in stand-by state and want to wake it up to copy files to its Backup directory.

1. In script Wake-on-LAN options (page 21) enable *Wake-on-LAN* and specify *IP address, MAC address* and *Port*.
2. Insert wait task (page 19) as first task in your script and specify `\\NAS\Backup` as value for option *Drive*.

Now tasks in script following your wait task are executed only after `\\NAS\Backup` has become accessible.

7.7 Possible annoyances

Possible annoyance due to architecture.

File not found errors

As all files that must be copied are detected before starting to copy first file it is possible that some files may have been deleted from source before they are actually copied, causing copy operation to fail. That is especially true if copy operation is time consuming -- the longer it takes to copy files the bigger is chance that some of the files needed to copy are deleted from source before they are actually copied.

Another similar problem arises when using shadow copy (VSS) (page 40). To avoid time and resource intensive task of creating volume shadow copy each time just to check if there were files to copy, files to copy are found using live disk instead of shadow copy. That allows to bypass creation of shadow copy if no files to copy were found. But if files to copy were found and then shadow copy is created, it is possible that shadow copy does not contain all the files that were initially found from live volume. That may be because some initially found files were excluded from shadow copy by shadow copy provider. Result is that file that existed on live volume may not exist on shadow volume, causing copy error. Usually files excluded from shadow copy are temporary files but nevertheless they cause unnecessary "failures".

To overcome these problems Copiaris logs "file not found" errors as simple notification, not error or warning, by default. If you want to to change that you need to change value of option *Log 'file not found' event as (page 27)*.

8 Advanced scripting

8.1 Advanced scripting

On most cases very simple script with one or few tasks that are executed one after another is sufficient. However, with Copiaris you can also use advanced scripting techniques like if-then-else blocks, custom variables and set task options from script to create advanced and conditional scripts.

Following topics cover all these features in detail.

8.2 If-then-else conditions

If-then

If-then-else is classic programming construct and it works by comparing value of existing user or built-in variable against specified value. If evaluation returns true then IF-block is executed, else not. If ELSE block is present then it is executed instead.

```
Run task 1 (any day)
If <DayOfWeek>=7 then
  Run task 2 (only on Sunday)
EndIf
```

The code above is pseudo code for script that contains 2 tasks and IF-block. Task 1 is executed every time the script is executed, but task 2 is only executed on Sundays. Here we have IF-block with condition `<DayOfWeek>=7` which means that task(s) inside IF-block are only executed when result returned by variable `<DayOfWeek>` equals 7 (Sunday). On any other days (`<DayOfWeek>` returns 1-6) IF-block is not executed.

If-then-else

We can also add ELSE statement to IF-block, which allows us to run one of two blocks, depending of IF-condition.

```
If <DayOfWeek>=7 then
  Run task 1 (only on Sunday)
Else
  Run task 2 (any other day than Sunday)
EndIf
```

Above we see pseudo code with ELSE block. In this example there is not task that is executed always, but task 1 is executed on Sunday and on any other days task 2 is executed.

Creating IF-blocks and specifying condition

To use IF-block you add it to your script like you add any task. Then specify condition by task options available in ***If statement*** section:

Variable `<variables>`

Name of the variable you want to compare. You can use any built-in or user variable name or even multiple variables combined. Result of these variables is compared against *Value*.

Condition

Comparison type. Condition *Contains/Does not contain* checks if variable contains or does not contain Value. Condition *File* *exists/File does not exist* assumes that *Variable* is file/directory name and checks if that exists or not. *Value* is not used on that case.

Value

Value you are comparing against. If left empty then you are comparing against empty string. Value is not used if condition is *File exists/File does not exist*.

Things to note

Comparing numbers with leading zeroes

Some variables can return numbers that have leading zeroes to keep file names or directories in proper sort order if these variables are used in file or directory names. For example `<Day>` variable returns `05` and not `5` if it is fifth day of the month.

If you want to use condition `if <Day>=5` then you can omit any leading zeroes and don't need to use `if <Day>=05` because Copiaris removes any leading zeroes on comparison if value is numeric (it converts numeric values to number when comparing).

Empty string is treated as zero

If you are comparing with numeric value and other member is empty then other member is treated like `0`. For example if you use `if <MyVar>=0` and user variable `MyVar` is empty then that condition equals true since empty variable is treated as `0`. Of course on this case you can also compare against empty string like `if <MyVar>=`

Comparison is not case sensitive

If comparing string then comparison is not case sensitive. That means if value of variable is 'Yes' and you compare against value 'yes' or 'YES' then they all are considered equal.

Sample scripts

Following sample scripts demonstrate how to use IF-blocks in real works situations:

Sample script: Find drive by label (page 54)

8.3 User variables

User variables are special kind of variables (page 35) that are defined in script by script creator using *Set variable task*.

Benefits of user variables

Can be assigned before running any task

Built-in variables are resolved when they are encountered in script (page 35), which usually does not cause any problem with typical variables that return system information, but Copiaris also supports several variables that prompt user to provide value at runtime, like select directory. So if you want user to enter any information at script runtime it is good idea to ask for information immediately when script is started, store returned value in user variable and then use that user variable later where required. Otherwise user may be prompted for information in middle of the script, causing unnecessary delays in processing.

Makes it easy to change same value of multiple options in one place

If you use same directory name in multiple places it may be good idea to assign it to user variable. For example if you have multiple copy tasks that copy to same destination then instead of setting value of *Destination directory* option for each task to `X:\Backup` you can define user variable `MyDest=X:\Backup` and then use `<MyDest>` as value for *Destination directory* of all tasks where you want to use same destination. That way you can also change your destination directory in one place quickly without digging in script and looking where you used hard coded directory name.

You can then also easily change `MyDest` user variable from hard coded directory name to one returned by variable, allowing user to select a directory at script runtime. For example if you just set user variable `MyDest=<BrowsePath>` then all tasks that use `<MyDest>` as destination will now copy to directory user selects when he/she starts script.

Can be used for advanced scripting

User variables in combination with IF-blocks (page 46) allow creating of advanced scripts that change behavior according to certain conditions. It is also possible to store task result in user variable (page 48).

Creating user variables

To define new user variable or change value of existing user value you add Set variable task (page 33).

8.4 Set option task

Set option task allows you to change task or script option at runtime from script. It is most useful if combined with IF-condition to change option value according to certain condition.

A very basic example of using set option task with IF-condition is the following, which disables copy task whose name is `CopyTask2` if script is running in console mode:

```
If <Console>='Yes' then
   Set option CopyTask2.Enabled='No'
EndIf
```

8.5 Storing task result in variable

Copiaris can store result of each task in user variable (page 47) and then you can check in your script using IF-block (page 46) if certain task was successful or failed.

The following pseudo code with IF-block checks if task copy task 1 was successful and only then executes task 2:

```
Run copy task 1 (and store task result to variable MyCopyTaskSuccess)
If <MyCopyTaskSuccess>='Yes' then
  Run task 2 (only if task 1 was successful)
EndIf
```

Assigning task result to variable

You can assign name of variable to *Assign result to variable* task option (page 22), then task result is stored in given variable. Task result is typically *Yes* if task was successful and *No* if it failed, but some task may offer more detailed result. For example show message task returns name of pressed button as result, so you know which button user pressed and can act accordingly. Example above used `MyTaskSuccess` as task result variable name. If variable with specified name does not already exist then it is created.

You can use different variable name for each task, then you have good overview in script of which tasks failed. Or, you can also use just one variable for multiple tasks but on that case value is only valid for latest task that has been finished and you have no means to check if certain earlier task was successful or not if more than one tasks have been already finished.

9 Command line parameters

9.1 Command line parameters

Copiaris has extensive support for command line parameters. You can even build entire script by using only command line parameters, without script file. All options you see in script editor can be used as command line parameters.

I refer `copiaris.exe` in examples below but you can specify different executable (⧉ page 51), depending if you want to run console or GUI mode (⧉ page 2). Usually you also need to include full path to exe file and put it between double quotes if it contains space. Full path and double quotes around `Copiaris.exe` are omitted here for clarity.

Referring options

If you want to work with command line parameters you need refer options by name to set their values. Options have long descriptive name that you see in script editor but also shorter name suitable for use in command line. See Referring options (⧉ page 51) on how to display option names in script editor.

Passing only script file name

Simplest use of command line parameters is passing script file to execute. Script file name must always be first parameter and be between double quotes and if script is not in the same directory as program executable then you must include full path:

```
Copiaris.exe "X:\Scripts\Full backup.copi"
```

Passing script file name and overriding some options

You can pass script to execute on command line and also specify additional options by command line, which results options loaded from script file being overridden by values passed by command line. Only values passed by command line are overridden, other values are loaded from script file. Currently you can override all script options (options you see in script editor when *Script* is selected), not options of tasks, but that is what you want to do on most cases since tasks can inherit options from script, so they also inherit new values that are assigned to script by command line.

Syntax:

```
Copiaris.exe "Script file.copi" [ScriptOption1=Value] [ScriptOptionN=Value]
```

Here we override value of *Zip compression password* script option by command line to avoid storing password in script file:

```
Copiaris.exe "X:\Scripts\Encrypt.copi" ZipPwd=Hst5SQku2
```

See How to pass encryption password by command line (⧉ page 53) for more details about this example.

Using without script file

One unique feature of Copiaris is that you can build entire script on the fly by using command line parameters. No ready-made script file is needed. Anything you can do using script file you can do from command line without using script file.

Syntax:

```
Copiaris.exe [ScriptOption1=Value] [ScriptOptionN=Value] [/TASKID] [TaskOption1=Value]
[TaskOptionN=Value] [/TASKID] [TaskOption1=Value] [TaskOptionN=Value]
```

where instead of `/TASKID` one of the following task ID-s is used (uppercase):

`/DEST`	copy task (⧉ page 16)
`/SRC`	copy source (⧉ page 16)

`/PROC`	process task (page 17)
`/WAIT`	wait task (page 19)
`/MSG`	show message task (page 18)
`/OPT`	set option task (page 48)
`/VAR`	set user variable task (page 47)
`/IF`	if statement (page 46)
`/ELSE`	start of else block
`/ENDIF`	end of if statement

Creating tasks and setting task options

If no script file name is passed as first parameter then Copiaris starts building new script from command line parameters. This kind of script is not stored on disk but is kept in memory and used for current session only. All parameters are parsed one after other and each time a task ID is encountered that type of task is added to the in-memory script and this task also becomes active task, which means any options that follow task ID must be options of that task.

The following example copies files from `D:\Files` to `X:\Backup\Files` by using VSS (page 40):

```
Copiaris.exe /DEST DestDir="X:\Backup\Files" /SRC SourceDir="D:\Files" UseVSS=Yes
```

What we see here is first task ID `/DEST` creates copy task, next parameter sets option `DestDir` (*Destination directory*) of copy task. Then following `/SRC` creates copy source to already added copy task. Last options assigns values to `SourceDir` and `UseVSS` options of copy source. As you can see copy task is before copy source also on command line, the same way as in script editor. The order of task parameters is very important as script is built in the same order as parameters are passed, so to add copy source you first must have added copy task, exactly the same way as in script editor. Once you have added copy task you can add multiple sources to it, if required. All other options that we didn't override remain default.

In script editor ***Tools | Copy entire script as command line*** menu command does what it says, it copies entire script file that is open in editor to the clipboard as command line (only enabled tasks and options with non-default value are included). That allows you to basically **build command line using script editor, visually**! Examine output of this command to learn how to use command line parameters and how to create tasks from command line.

Setting script options

Following example sends Wake-on-LAN (page 44) message to remote computer to wake it up from sleep state and exits.

```
Copiaris.exe WolEnabled=Yes WolIP=192.168.1.10 WolPort=9 WolMAC=001D73A44928 EndAction=Exit
/WAIT Timeout=1
```

If you need to set script options and also create tasks then you need to set all script options before creating first task. In example above, after setting wake-on-LAN script options, we also create wait task with 1 second timeout to prevent Copiaris from logging error that no tasks were executed. You can set any script options you see in script editor.

Important rules

- If option value contains space then you must double-quote entire option value, otherwise when command line is parsed it is not possible to detect parameters correctly. Example: `SourceDir="X:\Source files"`
- If option value contains double quote character " then that must be escaped by adding slash / before it, otherwise it breaks command line parameter parsing. Example: `UserVars="DestDir=<Drive label=/"My backup/">"`
- If you need to set script options and create also tasks then you need to set all script options before creating first task. Following example starts Copiaris in minimized window, launches calc.exe and closes Copiaris progress window: `Copiaris.exe Minimize=Yes EndAction=Exit /PROC ProcName=calc.exe Wait=No`
- Values of yes/no type of options can be set using either by *Yes|No* or *1|0.*
- When you have specified script file name as parameter 1 and also override options by command line then you currently can only only override root level script options (that you see in script editor when Script is selected in task list). Any use of task `/TASKID` parameters will result new tasks to be added to end of your script file loaded from disk as there is currently no way to refer existing task to override options of existing tasks.

9.2 Referring options

If you want to work with command line parameters or set value of option from script using set option task then you need refer options by name to set their values. Options have long descriptive name that you see in script editor but also shorter name suitable for use in command line.

For example script option *Confirm start message* must be referred by name *ConfirmStartMsg.*

To see short names of options choose ***Tools | Show command line syntax of options*** in script editor, then you see command line syntax of selected option in bottom left part of script editor.

9.3 Executable files

This documentation often refers to either `Copiaris.exe` or `CopiarisCon.exe` when showing examples of command line usage. On most cases you can substitute one to another, depending if you want to run standard or console mode (page 2) Copiaris. To run 64-bit console module use `CopiarisCon64.exe`.

Details of available executable files

`Copiaris.exe` - Script launcher that is started when you double-click script file. If you want to start script from another application or by *Run* applet then use this executable and pass script file and/or parameters on command line (page 49). This launcher invokes main `Copiaris32.exe` or `Copiaris64.exe`, depending on operating system in use.

`CopiarisCon.exe` - 32-bit console mode executable (page 2), run if you need to use 32-bit console mode Copiaris.

`CopiarisCon64.exe` - 64-bit console mode executable (page 2), run if you need to use 64-bit console mode Copiaris.

`Copiaris32.exe / Copiaris64.exe` - Main program modules that are invoked by launcher `Copiaris.exe` according to user operating system. Don't run these executables directly but always use `Copiaris.exe` that decides which of these program executables to start. This warrants that 64-bit executable is started on 64-bit Windows and 32-bit executable is started on 32-bit Windows. If you want to force 32-bit executable on 64-bit Windows you can run `Copiaris32.exe` directly but when running any of these `Copiaris32/64` execuables directly you are losing some functionality that is provided by launcher `Copiaris.exe`, like automatic elevation and script start prompt. Running these executables directly is not recommended and is not supported.

`CopiarisEditor.exe` - Script editor (page 11).

9.4 Exit codes

When calling Copiaris from batch files, automation tools or invoking it via process task (page 17) you can know if it finished successfully or issued warnings or errors by examining returned exit code.

Both GUI and console modules return the following exit code on program exit:

0 - success, no errors or warnings

1 - warnings

2 - errors

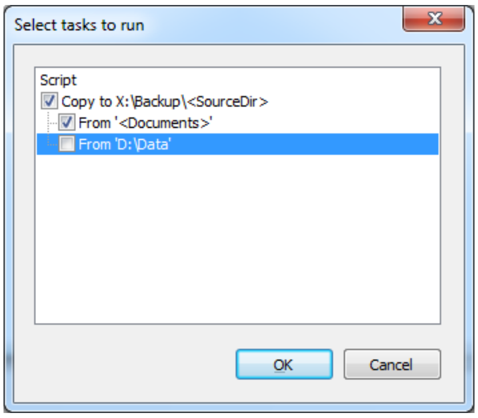
10 How to

10.1 How to select tasks to run at runtime

GUI only

Typically the one who designs script decides what tasks are executed by script and their order of execution. However, sometimes it may be better to allow script user to decide what tasks to run on that given time when he executes the script. For this purpose there is *Allow task selection at startup* script option (page 20).

If script option *Allow task selection at startup* = *Yes* then when script is started Copiaris displays window that lists all tasks from the script, including disabled tasks, and allows user to enable or disable any task, making it possible to choose which tasks are actually executed during that session. Tasks can also be re-ordered by dragging to change order of execution.



In sample above the script contains copy task with 2 sources. Here you can choose which tasks you want to run on that session. For example you may decide that today you don't want to copy files from source 2 (D:\Data) as you know it takes a lot of time and unselect this source. Then only files from source 1 (Documents) are copied now.

Any modifications to tasks order and enabled state are not saved to script file, they are used for current session only.

10.2 How to prompt for zip encryption password at runtime

If you want to use zip file with encryption then there is problem with storing encryption password that you can assign to *Zip encryption password* option. If you want to fully automate script so that it runs without user intervention then you must assign *Zip encryption password* and it is stored in script file. Although it is stored in scrambled form it is possible to reveal real plain text password by some reverse engineering. Unfortunately it is not possible to store only hash of user password because decompression program requires you to input same password you used for encryption, not hash of the password, so we need to encrypt with entered password and not hash of it.

There are a few ways to pass encryption password without storing it in script file.

Query encryption password at run time

GUI only

Copiaris allows you to query encryption password at run time, when user runs script. To do that you need to use variable (page 35) `<UserInput password>` as value in *Zip encryption password* option instead of your real password. Then Copiaris displays *Enter encryption password* window when running script to allow user to type in password at run time.

Using user variables

If you want to the same password in multiple tasks or have multiple time consuming tasks and zip task is not first task then you can use user variables (page 47) feature to first assign zip password to user variable and then use that user variable as value in *Zip encryption password* option, even for multiple tasks. This allows you to assign zip password at start of script, otherwise you will be prompted for password in middle of script when your zip task executes.

Pass encryption password as command line parameter

See Pass encryption password as command line parameter (page 53).

10.3 How to pass encryption password by command line

Here we override value of *Zip compression password* script option by command line to avoid storing password in script file. All tasks and other options are loaded from script.

First make sure that in *Zip encryption password* option of your copy task(s) is set to *Inherit*. Only that way all tasks inherit password that we set from command line since currently you can only override main script options from command line.

If you have multiple zip tasks and want to use encryption with only few of them then turn off *Inherit* of *Zip compression password* of these tasks, so blank password is used instead of inherited password and that disables encryption.

Following example passed script file name as first command line parameter and also assigns new password Hst5SQku2 to ZipPwd script option:

```
Copiaris.exe "x:\scripts\encrypt files.copi" ZipPwd=Hst5SQku2
```

Note: In this example we use `Copiaris.exe` to launch standard graphical interface but you can use different executable (page 51) to launch console module. Usually you also need to include full path to exe file and put it between double quotes if it contains space. Full path and double quotes around `Copiaris.exe` are omitted here.

10.4 How to check if task was executed

If you need to check in script if one previous task was executed or not use the following technique:

1. At start of script define user variable with no value, for example name it `MyTaskResult`.
2. Use same variable name `MyTaskResult` in your task option *Assign result to variable*, so if task runs task result is assigned to your variable. If task is not executed (it is disabled or skipped) then your variable remains empty.
3. Somewhere after the task use if-block in your script and check if your variable `MyTaskResult` is empty. If it is then the task was not executed, otherwise it contains task result value.

11 Sample scripts

11.1 Sample script: Find drive by label

Sample scripts are installed in sub-directory *Sample scripts* of your Copiaris installation directory.

File name of this sample script:

```
Find drive by label.copi
```

This sample asks user to insert disk with label "Backup" and then finds drive letter by label. If drive letter is found then files from My Documents are copied to that drive, else error message is displayed.

This sample shows how to:

1. Define and use user variables
2. Use if-then-else statements to do something only on certain condition(s)
3. Show message box with custom error message
4. Find drive letter of destination directory by disk label using `<drive>` variable

Things to note

1. We use "My" prefix for all user variables to distinguish them from built-in variables.
2. We use custom variable `MyDiskLabel` to hold label of disk. This makes it very easy to change label in one place if you use it in multiple places in script. In this example we use it as parameters to <Drive> variable and also in 2 message boxes.
3. The task that displays first message box with text *"Please connect backup drive..."* has both *OK* and *Cancel* button displayed by *Buttons* option. In addition under task options it has also *Terminate script on failure* set to *Yes*. That combination allows to terminate processing of script if user presses *Cancel* button in this message box.
4. Set variable task that sets value of `MyDriveLetter` variable has nothing defined for *Fail on* task option -- default value of that option, *error expanding value*, is unselected. That allows us to ignore potential error when expanding `<drive>` variable when no drive with required label is found. If *error expanding value* is selected in *Fail on* option then if we do not find drive with desired label error is logged but since we process error condition later by checking if `MyDriveLetter` is empty then we don't need logged error, thus we prevent it by removing *error expanding value* from *Fail on* actions.
5. In IF-block we have condition `<MyDriveLetter> <> ''` which causes block to be processed only if `MyDriveLetter` is not empty string, else custom error message is displayed.
6. We use drive letter we found by label as destination drive to copy files to as we use `<MyDriveLetter>\Backup` as destination directory. That way we don't need to hard code drive letter into script which is especially useful if we copy to removable drive that may get different drive letter assigned each time it is connected.

11.2 Sample script: Download photos from camera

Sample scripts are installed in sub-directory *Sample scripts* of your Copiaris installation directory.

File name of this sample script:

```
Download photos from camera.copi
```

This sample script allows you to download/copy photos from attached digital camera or memory card and rename files on the fly. Memory card or camera must be accessible via drive letter. This script renames copied files using YYYYMMDD_HHMMSS format, so resulting file names contain date and time of taking photos. That allows nice sorting of files by date even they are taken using different cameras and on different years.

How it works

1. The script tries to find camera or memory card reader drive letter by looking for drive that contains directory named "DCIM" and returns drive letter of first such drive found to user variable named `MySourceDrive`.
2. First IF-block checks if `MySourceDrive` received any drive letter. If it is still empty that means memory card is not connected. On that case prompt user to connect camera and then try once more to acquire drive letter that contains "DCIM" directory to same `MySourceDrive` variable.
3. Now start main block in case `MySourceDrive` contains drive letter (is not empty). In this block I have used several user variables to make the script more readable, but that makes script longer. Basically we ask user to enter name for sub-folder that is created for copied photos. This folder is created under *(My) Pictures* directory. We populate prompt with current date as default folder name, so user can leave it as is or he can type in his own text. Next is copy task that does actual copying. Things to note here are advanced source option *Merge sub-directories* is selected which causes photos from all digital camera folders to be merged into one destination folder, as camera usually makes additional folders inside main "DCIM" folder. Now actual renaming of files is instructed by source option *Rename by template*, that has file name template with variables `<FileDate>_<FileTime>.<FileExt>` specified, which results date and time of file to be inserted into file name. Since more than one photos can be taken on same second that template may result several photos having same file name which causes data loss if not handled, since directory can contain only one file having same name. To overcome this problem there is also script option *Rename if duplicate* selected, which causes preserving of duplicate files by appending unique number to the end of file name.
4. After copy task there is *Start process* task with command line `explorer.exe /n,"<MyDestDir>"` to open folder where photos were copied using Windows Explorer.

Typically users also want to delete copied photos from camera. In this sample script I have not enabled option to delete source files but if you wish to do so you can select advanced source option *Delete source files*. If that is selected then source files will be deleted after all files are successfully copied. If there was even one error during file copy then source files are not deleted for your own safety (that is by design of Copiaris).

As you can see Copiaris is quite flexible platform for this kind of work. You can easily set up a photo download script for someone who is not computer savvy. Just make shortcut to the script on Desktop and user can double-click it when he/she wants to download images from camera.

11.3 Sample script: Prompt for directory and zip

`GUI only`

Sample scripts are installed in sub-directory *Sample scripts* of your Copiaris installation directory.

File name of this sample script:

`Prompt for directory and zip.copi`

When this script is started it prompts user to select source directory and also destination zip file, then files from source are compressed to specified zip file. In addition it asks if zip file must be encrypted and if so user is prompted to enter encryption password. This sample shows that a script can be entirely interactive and source and destination can be specified at runtime, instead of being "locked" in script.

This sample shows how to:

1. Define and use user user variables
2. Display message box to user and later determine which button user pressed
3. Change script option if user pressed certain button in message box
4. Ask information from user at runtime and use it in script

Things to note:

1. We use "_" prefix for all user variables to distinguish them from built-in variables.
2. We ask source directory from user and assign it to variable `_SourceDir`.
3. We ask destination zip file name from user and assign it to variable `_ZipFile`. Note that `<BrowseFile>` variable is used and that you can specify default file extension and also default directory as parameters.
4. We ask if user wants to encrypt zip file. For message task we have specified `_EncryptZip` as value for *Store result to variable* option. That allows us to later know which button user pressed in message box and act accordingly. Show message task assigns name of pressed button to specified variable, contrary to most other tasks that return *Yes* on success and *No* on failure.
5. We use If-block to determine if user pressed *Yes* button (`if <_EncryptZip>=Yes`) and only on that case we assign new value to task option `ZipPwd`, which is result of <UserInput> variable, ie. the password/text user entered. For <UserInput> we pass additional parameters "password" and "confirm", these allow to enter and confirm password instead of plain text.
6. Copy task is executed, it uses variables `<_SourceDir>` and `<_ZipFile>` as source and destination, so copies from source that user selected when script was started.
7. We have specified our variable name `_ZipSuccess` in copy task's option *Assign result to variable* so that variable is set to *Yes* if compression is successful. We can check value of it and display error or success message.

11.4 Sample script: Wake sleeping NAS

Sample scripts are installed in sub-directory *Sample scripts* of your Copiaris installation directory.

File name of this sample script:

`Wake sleeping NAS.copi`

This script takes advantage of Wake-on-LAN (WOL) (↗ page 44) support in Copiaris and tries to wake up remote sleeping NAS (network attached storage device) or computer. You can use this kind of functionality in file copy script when you first need to make sure remote computer is not sleeping, but here we just use it to wake NAS and keep it wake until you press OK button to end script. While NAS is wake you can use any other programs to interact with it, like use Windows Explorer to browse and copy files etc.

- First you need to specify IP address, MAC address and port of your NAS or remote computer in *Wake-on-LAN* section of script options.
- We have *Wait* task in script that waits for remote drive `\\nas\backup` to be available before continuing with the script. You need to enter real name name and share of your NAS here. Wait task has also duration 300 seconds specified, that means it waits maximum 5 minutes for the drive to become available, after that script is terminated. If you need infinite wait enter 0 as duration.
- Last we have *Show message* task that notified user that NAS is accessible. Note that Copiaris sends WOL packets while script is running. That means sending of WOL packets ends when script is finished. Script is not finished until you press OK in displayed message, so while message is displayed on screen Copiaris keeps sending WOL packets on the background and all that time your NAS is accessible to all programs outside Copiaris.

11.5 Sample script: Code signing with SignTool

Sample scripts are installed in sub-directory *Sample scripts* of your Copiaris installation directory.

File name of this sample script:

`Code signing with SignTool.copi`

This calls external console program SignTool.exe from Windows SDK three times to add digital signature to 3 executable files that are in the same directory as script itself. Private key file for digital signature is on flash disk and password is asked at runtime. Drive that contains key file is found automatically by searching for drive that contains key file.

This sample is somewhat over engineered and can be done without IF-blocks but use of IF-blocks shows how to display nice messages to user on certain conditions.

How it works:

1. First we are looking for drive that contains *key.pk* private key file by setting `_KeyDriveLetter` user variable to `<Drive file="key.pk">`. Note that this task has *Terminate script on failure* set to *false* and also *Fail on* list is empty. That is because we don't want the task to report failure if it does not find required drive (when expanding variable fails) because we process that condition with next IF-block.
2. `If <_KeyDriveLetter> <> ''` is evaluated and returns true if our variable `_KeyDriveLetter` is not empty (drive with key file was found). On that case main if block is executed, else *Unable to find drive with key file* error message is displayed.
3. We set some user variables to make it easy to use same command line for all exe files.
4. We ask password from user at runtime and assign it to variable `_SignPwd`.
5. Process task is used to execute `SignTool.exe` with proper command line parameters, including password and key file. Note that under advanced process options we have set *Window state* to *Hidden*, so console window of SignTool is not shown, giving us better user experience. Any errors are still captured via process exit code.
6. After signing 3 executable files we use `<ErrorLevel>` variable in IF-block condition to determine if there were any errors or warnings logged. `<ErrorLevel>` returns 0 if there were no errors, 1 if there were warnings or 2 if there were errors logged during executing script.

Index

A

Advanced script options 20

Advanced scripting 46

Archive attributes 40

B

Built-in variables 37

C

Command line parameters 49

Common task options 22

Compression options 28

Console and graphical (GUI) mode 2

console mode 2

Copy source options 29

Copy task 16

Copy task options 23

Copying options 24

Creating your first backup script 9

D

Default options for copy tasks 22

Default options for e-mail 22

E

E-mail task 19

E-mail task options 33

Executable files 51

Exit codes 51

F

Features 1

File versions 13

Filtering options 26

full version 2

G

graphical mode 2

GUI mode 2

H

How to check if task was executed 53

How to pass encryption password by command line 53

How to prompt for zip encryption password at runtime 52

How to select tasks to run at runtime 52

I

If-then-else conditions 46

Include & exclude masks 41

Inheriting options 8

Introduction to variables 35

K

key file 5

L

Licensing 4

Limitations in free Lite version 2

lite version 2

Log file 13

Log file options 21

Logging options 27

M

Message from the author 3

Message task 18

Message task options 32

Migrating from Backup Magic 3

O

Optimizing for speed 44

Option value editor 36

Options 7

P

Possible annoyances 45

Process task 17

Process task options 30

Progress window 12

R

Referring options 51

Registry task 19

Renaming options 25

Running script 12

S

Sample script: Code signing with SignTool 56

Sample script: Download photos from camera 54

Sample script: Find drive by label 54

Sample script: Prompt for directory and zip 55

Sample script: Wake sleeping NAS 56

Scheduling scripts 43

Script 7

Script editor 11

Script options 20

Set option task 48

Set option task options 34

Set variable task options 33

Shell context menu 15

Storing task result in variable 48

System requirements 2

Z

Zip compression 15

T

Tasks 16

U

unlimited version 2

Unlocking Lite version 5

Unlocking with key file 5

User interface languages 3

User variables 47

W

Wait task 19

Wait task options 32

Wake-on-LAN 44

Wake-on-LAN options 21

V

Versioning options 27

Volume shadow copy (VSS) 40