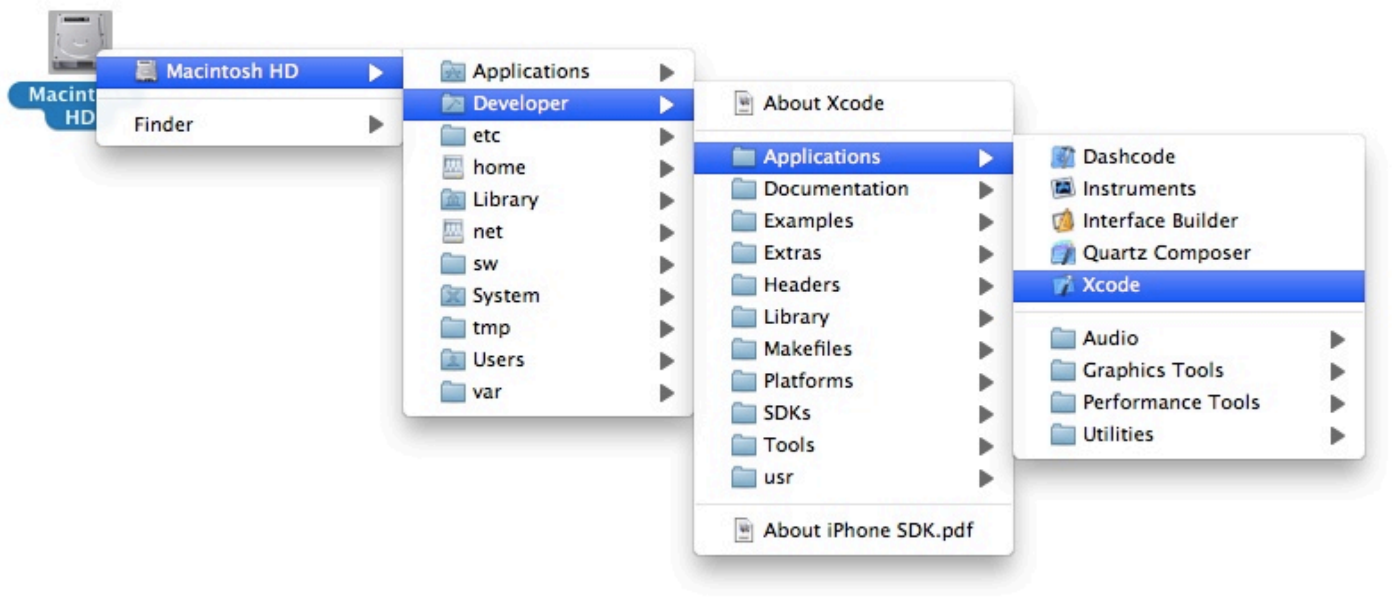


FolderGlance Scripting Guide

This guide covers FolderGlance 3.0 and later.
Last Updated: February 23. 2011.

1. A word of warning 3

Knowledge required 3

2. Where to place scripts 4

3. Types of scripts 4

4. Sample scripts 4

5. Naming conventions 4

6. User interface 6

7. Script parameters 6

8. Script output 7

9. Context Menu Interface (CMI) scripts 7

CMI script output format 8

10. Script lifetime 9

11. Debugging scripts 9

12. Improvements or suggestions? 10

1. A word of warning

FolderGlance Scripts provides a powerful way of extending the functionality of Folder-Glance. However – as they say – *with great power comes great responsibility.* Scripting is not for the faint of heart, and there are many risks you need to be aware of before you start delving into scripts. Scripts can do *anything*, including deleting files or folders, overwriting them, or otherwise corrupt them. Code wisely, and this won't happen. Further, be careful with what you do with the input your scripts get. As an example, consider a Python script that simply does the following:

```
#!/usr/bin/env python
import os, sys
os.system("echo "+sys.argv[1])
```

All this script does is run the shell's built-in "echo" command, printing its value. Seemingly innocous, right? No. This script is very dangerous. Let's say you have a file named as follows (without the quotes): "Dangerous; echo This could be dangerous"
If FolderGlance were to run the above script, the script's output would not be just the path to the file. Instead, due to the semantics of the "os.system()" call and the shell, it would actually *execute part of the file's filename!* The scripts output would be something like the following:

```
/Users/daniels/Dangerous
This could be dangerous
```

Note how the semi-colon went missing, along with the tail end of the filename. Needless to say, if the name of the file or folder contained "rm -rf /*" instead of the aforementioned echo command, you would suddenly find yourself without any files on your harddrive.

FolderGlance does not attempt to sanitize input to scripts – that is the script's job. So be careful with what you do within your scripts; seemingly safe calls may well turn out to be unsafe. Yellow Lemon Software takes no responsibility for damage caused to your Mac through the use of FolderGlance or any scripts FolderGlance executes.

With these words of warning in place, please read on to discover how useful scripts can be and how to write them for use with FolderGlance.

1.1. Knowledge required

To successfully write scripts for use with FolderGlance, you should have some programming experience and be familiar with Terminal. I can not provide assistance with programming scripts, although I will help you with specific questions regarding the interface between FolderGlance and your script.

2. Where to place scripts

FolderGlance locates its scripts in the FolderGlance Scripts folder. The location of this folder defaults to ~/Library/Application Support/FolderGlance/Scripts/, but can be configured in the Custom Folders section of FolderGlance's preference pane. Any scripts to be executed by FolderGlance must be placed in the folder configured here. Note that when you update FolderGlance, the installer will create a backup of the existing scripts folder and overwrite any scripts that ship with FolderGlance with new versions.

3. Types of scripts

Scripts to be executed by FolderGlance must be either marked as executable or have an executable placed in an application bundle. Symbolic links or aliases to executables or application bundles also works. Assuming you have written a script called "my-script.py," you can mark the script as executable by executing the following command in Terminal:

```
  chmod a+x <path to script>/my-script.py
```

At this point, you should be able to execute the script:

```
  cd <path to script>
  ./my-script.py
```

FolderGlance will never execute your script without any arguments. Scripts can be written in any language, as long as you have an interpreter on your system that can execute the script. You can also compile your own binaries from C/C++/Objective-C code (or whatever language you prefer to code in) and use them as scripts.

4. Sample scripts

FolderGlance ships with a number of sample scripts. These scripts are located on the FolderGlance disk image, in the "Sample scripts" folder. The sample scripts provided are:

`File and folder count (selection, implicit).py`	A simple script written in Python that simply counts the number of files, folders and symbolic links in the current selection. The script is executed implicitly when the context menu is opened. This script is also installed by default when FolderGlance is installed.
`Disk Usage (selection, implicit)`	This is a compiled executable. The source code is also provided. This script is installed by default when FolderGlance is installed.
`Sample dynamic script (selection, cm-interface).py`	This script demonstrates how you can build your own hierarchical menu structure. The script is written in Python. This script is *not* installed by default, as it doesn't do anything useful except to demontrate the mechanism for creating menu hierarchies and responding to user selections.

5. Naming conventions

Scripts can be named in almost any way. A script's extension does not impact how it is executed. Special keywords in a script's name impact the way it is executed. FolderGlance looks for keywords in between a set of parentheses in the script's name. These non-case sensitive keywords are:

`selection`	This script can only be applied to the current selection.
`implicit`	This script is executed implicitly when the context menu is opened. When combined with the `selection` keyword, this script is passed the selection as its parameters, otherwise it is passed the parent folder of the selection as its only parameter. **Note: Providing "implicit" without "selection" is currently unsupported.**
`cm-interface`	This script supports the FolderGlance Context Menu interface. When this keyword is present, the script is executed implicitly to generate a menu hierarchy presented by FolderGlance in the context menu. As with the `implicit` keyword, this keyword can be combined with the `selection` keyword to control whether it receives the current selection or the parent directory of the current selection as its argument(s). **Note: Providing "cm-interface" without "selection" is currently unsupported.**
`disabled`	This script will not appear in FolderGlance's menus.

The keywords can be combined by separating each keyword with a comma, although some combinations (or lack thereof) are currently unsupported. Some example script names and their effect are shown below:

`Sample script.py`	This script does not have any keywords in its name. When executed, this script will receive a single parameter, which will be the parent directory a user has browsed to. When executed from the root context menu, this script will receive the parent directory of the current selection. If the user presses the Option-key, the script when executed from the root context menu will receive the current selection as its parameters instead.
`Sample script 2 (selection).py`	This script will only appear in the root context menu, and it will only receive the current selection as its parameter. Pressing the Option-key while the context menu is open does not change the parameters passed to the script.
`Sample script 3 (implicit, selection).py`	When the user opens the context menu, this script will be executed immediately. The final line of output from the script will be used as the name of the menu item when the script is done executing. It will receive the absolute paths of the current selection as its arguments.

`Sample script 4 (implicit).py`	**This keyword combination is not yet supported.** *Intended behaviour:* As with Sample script 3, this script will execute when the context menu is opened. It will receive the absolute path to the parent directory of the selection or folder you browse to, and the last line of output from the script will be used as the menu item's title in the context menu.
`Sample script 5 (cm-interface, selection).py`	This script executes when the context menu is opened. Each line of output describes a single menu item which can be selected by the user, and must follow a specific format. See the section on Context Menu Interface scripts. The script receives either --populate or --select <menu item id> as its first argument(s), followed by the current selection.
`Sample script 6 (cm-interface).py`	**This keyword combination is not yet supported.** *Intended behaviour:* This script behaves like Sample script 5, except that its only argument is the parent directory of the current selection.

Note that the keywords and their associated parentheses are filtered from the script name when the name is shown in the context menu. Some of the combinations are not yet supported, and may produce unexpected behaviour.

6. User interface

All scripts, with the exception of scripts tagged with the `cm-interface` keyword, produce a single menu item in FolderGlance's root context menu. Scripts without the `selection` or `implicit` keywords applied will also have a menu item appear at the top of the menu in folders you browse to. The title of these menu items is the name of the script, sans any keywords and their associated parentheses. When the user selects a menu item associated with your script, your script is executed with the relevant parameters; either the absolute paths to the current selection, or a path to the directory being browsed to.

If the script is tagged with the `implicit` keyword, the script is executed immediately when the context menu is opened. Once the script has finished executing, the menu item's title is changed to the last line of the script's output, and the menu item is disabled (since it has already executed).

Note that currently, implicitly executed scripts that do not apply to the selection are only executed for the parent folder of the current selection, and not for any folders you browse to.

7. Script parameters

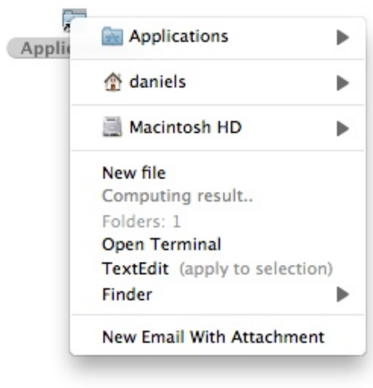
In general, all scripts receive absolute paths as their parameters. Scripts typically operate on either the selection or a single directory. When operating on the selection, the parameters will contain at least one path to either a file or folder, and possibly several such paths. If a path points to a symbolic link or alias, the script will have to resolve the link or alias itself.

The exception to this is for scripts with the `cm-interface` keyword in their names. See the section on Context Menu Interface (CMI) scripts for more information on the parameters to CMI scripts.

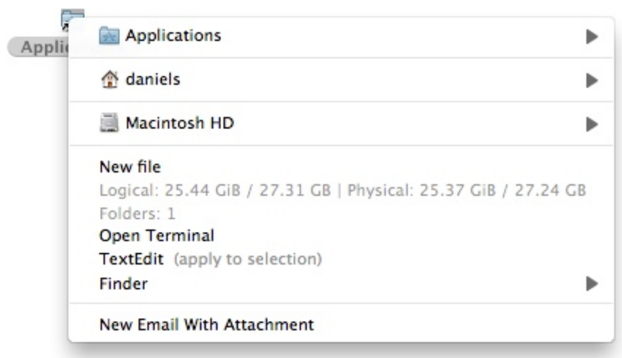
8. Script output

FolderGlance only uses output from scripts that are executed implicitly. To execute a script implicitly, the script's name must contain either the `implicit` or `cm-interface` keyword. Output should be sent to standard out.

Scripts with the `implicit` keyword must print at least one line to standard out. The last line printed by a script will be used by FolderGlance to replace the title of the script's menu item. While the script is executing, FolderGlance will show "Computing result…" as the title of the script's menu item. The title is updated when the script finishes executing. For the expected output from CMI scripts, see the section on CMI scripts.



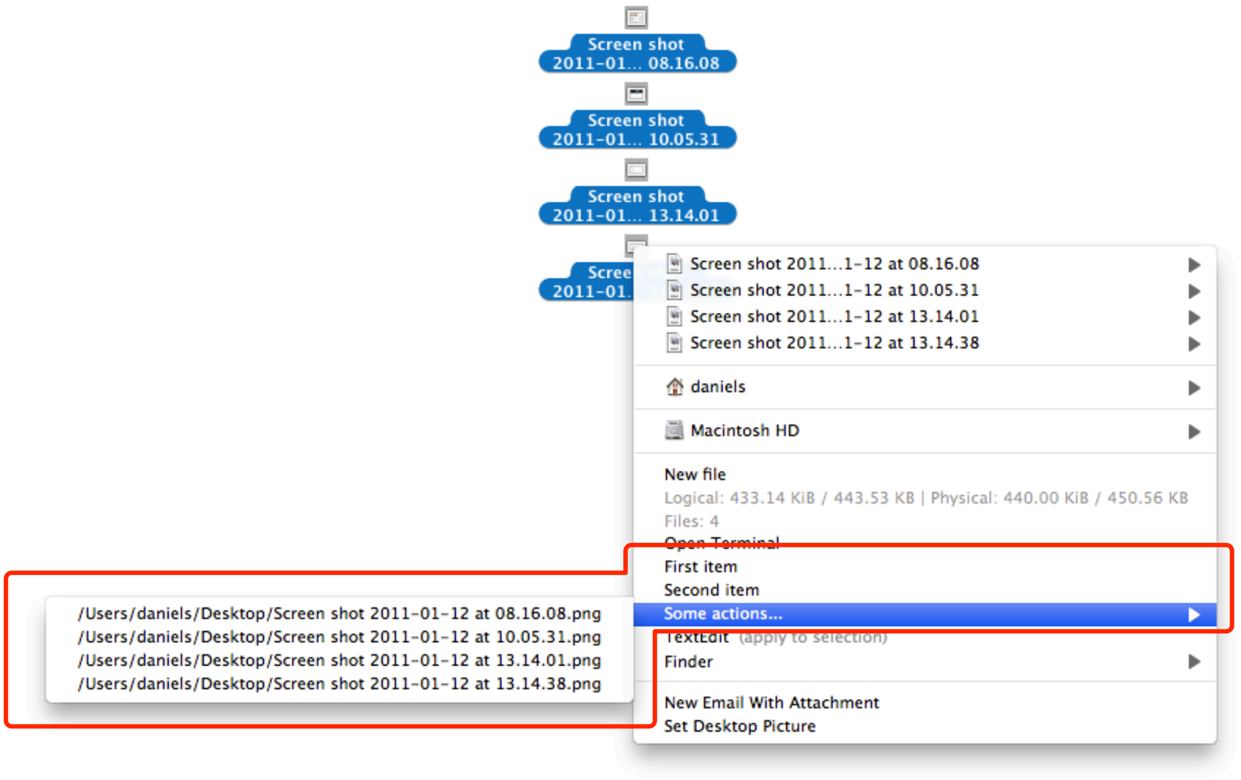
1. The disk usage script is computing the size of the Applications folder


Appli
Applications
daniels
Macintosh HD
New file
Logical: 25.44 GiB / 27.31 GB | Physical: 25.37 GiB / 27.24 GB
Folders: 1
Open Terminal
TextEdit (apply to selection)
Finder
New Email With Attachment

2. After a short while, the script is done executing, and "Computing result…" is replaced with the script's output.

9. Context Menu Interface (CMI) scripts

Context Menu Interface scripts are different from regular FolderGlance scripts in that they are expected to produce a menu structure that FolderGlance will interpret and use to generate a menu hierarchy. An example of such a hierarchy is shown in the screenshot below. The red highlight indicates menu items generated by the script.


Screen shot 2011-01... 08.16.08
Screen shot 2011-01... 10.05.31
Screen shot 2011-01... 13.14.01
Screen shot 2011-01...
Screen shot 2011...1-12 at 08.16.08
Screen shot 2011...1-12 at 10.05.31
Screen shot 2011...1-12 at 13.14.01
Screen shot 2011...1-12 at 13.14.38
daniels
Macintosh HD
New file
Logical: 433.14 KiB / 443.53 KB | Physical: 440.00 KiB / 450.56 KB
Files: 4
Open Terminal
First item
Second item
Some actions...
TextEdit (apply to selection)
Finder
New Email With Attachment
Set Desktop Picture
/Users/daniels/Desktop/Screen shot 2011-01-12 at 08.16.08.png
/Users/daniels/Desktop/Screen shot 2011-01-12 at 10.05.31.png
/Users/daniels/Desktop/Screen shot 2011-01-12 at 13.14.01.png
/Users/daniels/Desktop/Screen shot 2011-01-12 at 13.14.38.png

To accomplish this, CMI scripts are executed twice: First, when the context menu is opened, and second only if the user selects one of the menu items produced by the script. The first time a CMI script is executed, the first argument to the script will be `--populate`. On the second invocation, the script's two first arguments will be `--select <menu-item-id>`. The remaining arguments (paths to files and folders) will remain constant in between the two invocations. To summarize, a CMI script is conceptually invoked as follows:

```
  ./my-cmi-script --populate <path1 [path2 ...]>
```

or

```
  ./my-cmi-script --select <menu-item-id> <path1 [path2 ...]>
```

Since the script doesn't keep running in between the first and second invocation, it needs to generate a set of menu item IDs based on the parameters it receives. These menu item IDs are sent back to FolderGlance, along with the title of the menu item. When a user selects an item generated by your script, it is invoked again, this time with `--select <menu-item-id>` as its first two arguments. Note that the menu item IDs you generate should be unique. If you supply two items that both use the same menu item ID, the behaviour of your script will probably not be as expected.

9.1. CMI script output format

The following specifies how your script should print its output in order to generate a menu structure:

```
  0
  [tabs] <title> <tab> <unique-id> <tab> <flags>
  ...more items following above specification, one per line
```
The first line of output should be the numeral zero (“0”), followed by a newline. This indicates to FolderGlance which version of the output format is in use; currently only version 0 is defined. Every line following this line describes a single menu item. Each line is divided into fields, using the tab-character to separate different fields.

The first field is the menu item’s title. The next field is the menu item’s unique ID, which should be a 32-bit literal integer. The final item is the menu item’s flags. At present, flags are not used by FolderGlance, and this field should be set to 0.

Before the first field, zero or more tab-characters may be present. These indicate the submenu level that the item being described should appear at. It is an error to nest an item more than one level below the level above it.

The following example demonstrates the output to generate six items, where the last items appear in sub- and subsubmenus (similar to the screenshot shown above). For clarity, the actual tab-character has been replaced with the text <TAB>. The script’s command is also shown with a dummy path:
```
  ./my-cmi-script --populate /
  0
  My first item<TAB>0<TAB>0
  My second item<TAB>1<TAB>0
  My first submenu<TAB>2<TAB>0
  <TAB>My first submenu item<TAB>3<TAB>0
  <TAB>Second submenu item, with submenu<TAB>4<TAB>0
  <TAB><TAB>Subsubmenu item<TAB>5<TAB>0
```
If the user then selects for instance item four, “My first submenu item,” the script will be executed again as follows:
```
  ./my-cmi-script --select 3 /
```

To learn more about writing a CMI script, look at the sample script called “Sample dynamic script (selection, cm-interface).py”. This script is located on the FolderGlance disk image, in the Sample scripts folder.

10. Script lifetime

Scripts executed by FolderGlance are not terminated. If you need to cancel a script, you will need to use for instance Activity Monitor to locate the script’s process and force quit it from there.

In the future, scripts tagged with the `implicit` keyword will be terminated once the context menu is closed, however no such restriction is in place for scripts executed in response to the user selecting a script’s menu item.

11. Debugging scripts

In general, it is best to debug your scripts by running them manually from Terminal and inspect their output. Only once you are confident that you get the results you expect, should you place the script in FolderGlance’s scripts folder. There are two further techniques you can use for debugging: Start the Finder in Terminal, and enable FolderGlance's debug output.

To start the Finder in Terminal, execute the following command in Terminal (it is all one line):

```
  killall -9 Finder &&
  /System/Library/CoreServices/Finder.app/Contents/MacOS/Finder
```

This will enable you to see information the script sends to standard error as they run. Alternatively, you can terminate the Finder more gracefully as follows:

```
  osascript -e 'tell app "Finder" to quit' &&
  /System/Library/CoreServices/Finder.app/Contents/MacOS/Finder
```

You can also enable debug output in FolderGlance. Be warned that FolderGlance may print a lot of information, and little of it is related to scripts. To enable debug output, open System Preferences. Next, while holding the Option (Alt) key down, open the FolderGlance preference pane and release the Option key. Select the Advanced section, in which there should now be a popup that allows you to turn logging off, send log output to standard out (good for when you run the Finder from Terminal), or to a file. If you choose to log to file, output will be stored in ~/Library/Logs/FolderGlance.log.

12. Improvements or suggestions?

If you have any suggestions on how this guide could be improved, or how FolderGlance's scripting support could be improved, please feel free to send me an e-mail with your comments: yls@scsc.no.