<div align="center">

**Eagle Mode -**

# emFileMan Customization

</div>

# Contents

# 1 Introduction

This document describes how to customize the commands of the Eagle Mode File Manager. The commands are made of small script files. On Linux/UNIX, these files are written in the Perl programming language, and on Windows, they are written in Javascript. But for simple modifications, you do not need to know anything about these languages.

The script files are located in the directory `$EM_DIR/etc/emFileMan/Commands` (replace `$EM_DIR` by the installation directory of Eagle Mode, and replace etc by etcw if on Windows). Commands of subgroups are located in corresponding subdirectories. You could modify the commands directly there, but this may require administrator privileges. Alternatively, you could create a copy of the whole directory tree, so that it is possible to modify the files with user privileges. The required path for that copy is `$HOME/.eaglemode/emFileMan/Commands` (or on Windows: `%APPDATA%\eaglemode\emFileMan\Commands`). If that directory exists and seems okay, the script files are loaded from there, and the other is ignored.

Adding a new command simply consists of adding a new script file. To make a change visible to Eagle Mode, trigger the button *Reload Files*. Error messages for bad script files are printed to STDERR, and therefore it is a good idea to run Eagle Mode from a terminal while working on the commands (exception: on Windows the messages are written to `%TMP%\emCoreBasedAppLog.log`).

Remember to make backups of your modifications, because they could be removed when reinstalling Eagle Mode.

# 2 File types and applications

This chapter is about modifying the mapping of file types to applications. But that is described only for Linux/UNIX here. There, Eagle Mode has one file manager command for each application. On Windows, there is just one command for all: "Open or Run". It is the default action for all files and it uses the mapping from the operating system, so that you can configure there (see Windows Control Panel -> Default Programs).

Yes, it would be possible to create such an "Open or Run" command also on Linux/UNIX, which forwards to a central default program service - if you have one.

## 2.1 Setting the preferred application for a file type

For example, if you want to make Kate the default text editor, edit the file `PlainText/Kate.pl` (below the *Commands* directory described above) and find the following line in it:

```
# Order = 2.0
```

Simply set the order number to zero:

```
# Order = 0.0
```

Thereby, Kate gets precedence over all the other text editors. And besides, the button moves to the beginning of the group.

## 2.2 Adding a new application for a file type

Assumed you have an application that can edit or show files of a certain file type, and you want to be able to run it out of the file manager. Then you have to create a new file in the desired group directory. Give it a name that ends with `.pl` and fill it with this template:

```
#!/usr/bin/perl
#[[BEGIN PROPERTIES]]
# Type = Command
# Order = 0.0
# Interpreter = perl
# DefaultFor = type
# Caption = name
#[[END PROPERTIES]]

use strict;
use warnings;
BEGIN { require "$ENV{'EM_DIR'}/res/emFileMan/scripts/cmd-util.pl"; }

OpenSingleTargetFileWith('program');
```

And then replace three things in the template:

| type | Replace this by the file type the application is able to handle. It must be a file name ending including the leading dot (something like `.avi`), or a colon-separated list of such (e.g. `.avi:.mpg`), or the special keyword `file` for the case the application is a plain text editor. |
|---|---|
| name | Replace this by the application name. It is shown in the button. |
| program | Replace this by the executable file name of the application, like it is called in a shell, with or without path. |

If you want to be able to open multiple files at once, and if supported by the application, you could replace, in the last line of the script, the word `OpenSingleTargetFileWith` by `OpenTargetFilesWith`.

# 3 Reference

## 3.1 Properties of commands and groups

Each command script file has a special section containing properties. And for each command group directory, there is a file which even has such a section. The properties are configuration parameters read and used by the file manager.

The format of the properties section is:

```
#[[BEGIN PROPERTIES]]
# name = value
# name = value
# name = value
...
#[[END PROPERTIES]]
```

On Windows, i.e. with Javascript, the section has to be enclosed in a Javascript comment (/*...*/).

Possible properties are:

| Name | Description |
|---|---|
| Type | Type of the element. Possible values are:<br>  Command  - The file is a command script.<br>  Group     - The file describes a command group.<br>This property must always be there. |
| Order | Order of the element. This can be any number. Within a group, the elements are sorted by these numbers. The element with the lowest number is at the beginning. |
| Interpreter | This property is only for commands. It specifies the program to be called as the interpreter for the script. For Perl scripts, say perl. |
| Directory | This property is only for groups, and there it is always required. It specifies the name (without path) of a subdirectory containing the elements of the group. |
| DefaultFor | This property is only for commands. It specifies the file types for which the command is a default command. The value for this property must be one of:<br> * A colon-separated list of file name endings, each including the leading dot.<br> * The keyword file. This means to match all regular files.<br> * The keyword directory. This means to match directories. |
| Caption | A caption to be shown in the label of the button or group. |
| Description<br>or Descr | A description to be shown in the label of the button or group. This property can be given multiple times for creating a multi-line description. |
| Icon | An icon to be shown in the label of the button or group. It can be specified by a file name of a TGA file in the directory $EM_DIR/res/icons. A path to a TGA file is also allowed, either relative to that directory, or absolute. |
| BgColor<br>FgColor<br>ButtonBgColor<br>ButtonFgColor | Background and foreground colors of the borders and buttons. The colors of a group are used as the default for the elements of the group. A color can be specified as a color name like Powder Blue, or as a hexadecimal RGB value like #B0E0E6 or #BEE. |

| | |
|---|---|
| `Hotkey` | This property specifies a hotkey for the command. Sample values are: `Ctrl+C`, `Shift+Alt+D`, `Meta+F10` |
| `BorderScaling` | Scale factor for the size of the border. The default is 1.0. Zero means to make a group panel unfocusable. |
| `PrefChildTallness` | Preferred height/width ratio of the elements of the group. The default is 1.0. |

## 3.2 Precedence of default commands

By setting the `DefaultFor` property, you can make a command the default command for a file type. But if multiple commands are the default for the same file, there is a conflict. In that case, the file manager chooses a command by the following rules (strongest-first):

- If a command A matches the file by file name ending and a command B matches just by matching every file (keyword `file`), then A has precedence over B.

- If a command A matches the file by a longer file name ending than a command B (e.g. `.tar.gz` vs. `.gz`), then A has precedence over B.

- If a command A lies in a parent group (or ancestor group) of the group of a command B, then A has precedence over B.

- If a command A has a lower `Order` number than a command B, then A has precedence over B.

The rest is tree sorting by captions, case-sensitive.

## 3.3 Utility functions

At the beginning of each command script, there is some code which loads and executes a framework. It is the file `$EM_DIR/res/emFileMan/scripts/cmd-util.pl` on Linux/UNIX, and `%EM_DIR%\res\emFileMan\scripts\cmd-util.js` on Windows. This performs some initializations and defines a lot of of utility functions. For information about these functions, please have a look into that file, and take all the existing command scripts as usage examples.

The rest of this document may be of interest only for understanding the implementation of the framework, or for writing a new framework in another scripting language.

## 3.4 Arguments

The file manager calls the command scripts with certain arguments: The first argument is the number of source-selected files and the second argument is the number of target-selected files. Then there are the absolute paths of the selected files, with one argument per file, and with the source-selected files first.

## 3.5 Environment variables

When the file manager calls a command script, following environment variables are set in addition to normal variables:

| Name | Description |
|---|---|

| | |
|---|---|
| `EM_DIR` | Directory where Eagle Mode has been installed. |
| `LD_LIBRARY_PATH` or `PATH` or `DYLD_LIBRARY_PATH` (depends on OS) | Contains $EM_DIR/lib, for that the libraries can be found. |
| `EM_FM_SERVER_NAME` | Name of the file manager's emMiniIpc server. |
| `EM_COMMAND_RUN_ID` | An identification for this run of a command. |
| `EM_X`<br>`EM_Y`<br>`EM_WIDTH`<br>`EM_HEIGHT` | Position and size of the application window. This can be used to calculate a good position for popup dialogs. |
| `EM_ACP` | Only on Windows: ANSI code page. |

## 3.6 The file manager's emMiniIpc server

A command script can send certain requests to the file manager. For doing that, the script can execute the program `emSendMiniIpc` in the directory `$EM_DIR/bin`. The first argument must be the server name (say `$EM_FM_SERVER_NAME`), and all further arguments are making up the request, which can be one of these:

`update`
> Require the file manager to reload all the shown files and directories (mostly only if the file modification time has really changed).

`select` *<command-run-id>* [*<file>* [*<file>* ...]]
> Like `update`, but also require the file manager to target-select the given files, or to clear the target selection if no files are given. As usual, the source selection is set to the old target selection. The *<command-run-id>* must be set to the value given with the environment variable `EM_COMMAND_RUN_ID`. The sense of that id is: If the user has modified the selection since the file manager command has been started, or if another command has been started afterwards, the selection request is ignored automatically.

`selectks` *<command-run-id>* [*<file>* [*<file>* ...]]
> Like above, but the source selection is not changed (ks = keep source).

`selectcs` *<command-run-id>* [*<file>* [*<file>* ...]]
> Like above, but the source selection is cleared (cs = clear source).

---

Next Reading: <u>Advanced Configuration</u>