Eagle Mode -
Advanced Configuration

Contents

1 Font hacking

The fonts of Eagle Mode are defined in TGA files. The original files do not cover the whole set of Unicode characters. If you want to add or replace some, then you could either paint them yourself, or you could convert standard font files with the font2em program. Of course, one day Eagle Mode should be able to use standard font files directly.

Read here how the TGA files are organized and how you can extend the character set by conversion from standard font files.

1.1 Important hints

First of all, note some important hints on the font hacking:

- Only mono spaced fonts (aka fixed fonts) are suitable for conversion.
- font2em is a less-than-ideal solution and it lacks some options. Especially with vector fonts, it may cut off something from extended characters, or it may choose a too large character box.
- You must make sure that the TGA files do not have too large dimensions, otherwise the font cache of Eagle Mode may not be able to hold the fonts at all, or the font cache may load and unload them too often. Especially CJK sets are problematic. If Eagle Mode often displays grey "Costly" boxes instead of correct characters, then you have a problem here and you should recreate your TGA files with a smaller character pixel height. Another chance would be to increase the maximum size of the font cache, but currently that would have to be made in the source code of Eagle Mode: It's at the end of `$EM_DIR/src/emCore/emFontCache.cpp`, the variable named `emFontCache::MaxMegabytes`.

1.2 The TGA files

The TGA files lie in `$EM_DIR/res/emCore/font`. Each file defines another range of Unicode characters. Therefore, the names of the TGA files have the following form:

<first>`-`*<last>*`_<`*width*`>x<`*height*`>_<`*desc1*`>_<`*desc2*`>.tga`

with:

<*first*> - Hexadecimal Unicode of first defined character.
<*last*> - Hexadecimal Unicode of last defined character.
<*width*> - Decimal character width in pixels.
<*height*> - Decimal character height in pixels.
<*desc1*> - Description of the Unicode range.
<*desc2*> - Additional description ("original" means painted by me).

There are two additional files, which are always required: CostlyChar.tga and UnknownChar.tga. They are outside Unicode.

All characters of a file have the same size. Each file contains the characters row-by-row. The number of columns is detected automatically.

If files overlap in the range of Unicodes, the behavior is undefined.

1.3 font2em

font2em is a small kludgy tool which can be used to convert standard font files (TTF, PCF, PFB,...) to TGA files for use in Eagle Mode. It is based on the freetype library. Here comes an example of how to convert some Cyrillic and Japanese characters:

```
$EM_DIR/bin/font2em /usr/share/fonts/truetype/DejaVuSansMono-Bold.ttf /tmp/deja 112
$EM_DIR/bin/font2em /usr/share/fonts/misc/18x18ja.pcf.gz /tmp/jap 18
```

The first command creates a `/tmp/deja` directory with some TGA files which are converted from the DejaVuSansMono-Bold font. The pixel height of characters gets 112 (the width is set automatically). The second command creates `/tmp/jap` from a Japanese font. There the height argument is just used for choosing a nearest possible height, because the source font is already a bitmap font.

Have a look at the files in `/tmp/deja` and `/tmp/jap`. Search for TGA files containing characters you want to install, and install them by copying to the font directory:

```
cp /tmp/deja/*Cyrillic*.tga $EM_DIR/res/emCore/font
cp /tmp/jap/*CJK*.tga $EM_DIR/res/emCore/font
cp /tmp/jap/*Hiragana*.tga $EM_DIR/res/emCore/font
cp /tmp/jap/*Katakana*.tga $EM_DIR/res/emCore/font
```

Finally remove the temporary directories:

```
rm -r /tmp/deja /tmp/jap
```

1.4 font2em as an emTmpConv plugin

Privately, I use font2em also for viewing standard font files in Eagle Mode. But it's not very nice, and beginners could think TTF files would consist of TGA images. Anyway, if you want it (please not in distributions), here is the contents of my plugin file (name it `emTmpConv_font2em.emFpPlugin` and put it into `$EM_DIR/etc/emCore/FpPlugins`).

```
#%rec:emFpPlugin%#

FileTypes = { ".otf" ".pcf" ".pcf.gz" ".pfa" ".pfb" ".ttf" }
Priority = 0.51
Library = "emTmpConv"
Function = "emTmpConvFpPluginFunc"
Properties = {
 {
  Name="OutFileEnding"
  Value=".dir"
 }
 {
  Name="Command"
  Value="exec \"$EM_DIR/bin/font2em\" \"$INFILE\" \"$OUTFILE\" 32"
 }
}
```

2 Plugin configuration

2.1 emFpPlugin - file panel plugins

The main plugin interface of Eagle Mode is named emFpPlugin (Eagle Mode File Panel Plugin). Such a plugin is responsible for showing the contents of a file in Eagle Mode. There are configuration files that tell which plugins have to be used for which file types. And because some file types can be shown by multiple plugins, there is a priority mechanism by which the plugins are sorted. In addition, some plugins have special parameters ("properties") which can also be configured.

By default, the configuration files are in the directory `$EM_DIR/etc/emCore/FpPlugins` (replace $EM_DIR by the installation directory of Eagle Mode, and replace etc by etcw if on Windows). But you can also create a copy of that directory, so that it is possible to modify the files without administrator privileges. The required path for the copy is `$HOME/.eaglemode/emCore/FpPlugins` (on Windows: `%APPDATA%\eaglemode\emCore\FpPlugins`). If that directory exists and seems okay, the configuration files are loaded from there, and the other is ignored.

For being recognized by the program, the names of the configuration files must end with ".emFpPlugin".

Here comes a description of the file format in form of a commented example for an imaginative plugin:

```
#%rec:emFpPlugin%#
  # Must be the first line of the file.

FileTypes = { ".cherry" ".pear" ".banana" }
  # Any number of file types the plugin is able to show. Each entry must be
  # either a file name ending including the leading dot, or "file" which means
  # all regular file types, or "directory".

Priority = 1.0
```

```
  # Priority for the plugin. This is important when there are multiple plugins
  # able to show a file type. Then, the plugin with the highest priority is used
  # as the primary viewer, and the others are used as alternative viewers in an
  # order according to decreasing priority. The priority can be any number.
  # Typical priorities are:
  #  1.0 - Specialized file viewer
  #  0.5 - Temporary conversion (emTmpConv)
  #  0.1 - Plain text viewer
  #  0.0 - Hex dump

Library = "Fruit"
  # Name of a dynamic link library making up the plugin. The name must be given
  # without path, without ending and without the "lib" in front (e.g. say
  # "Fruit" instead of "/usr/local/eaglemode/lib/libFruit.so").

Function = "FruitFpPluginFunc"
  # Symbol name of a C function in the library which creates the panel for a
  # given file. The exact function declaration can be found in the header file
  # "emFpPlugin.h".

Properties = { { Name="color" Value="blue" } { Name="size" Value="7" } }
  # This is optional. It is a list of additional configuration parameters in
  # form of name/value pairs. Both, the names and the values are always quoted
  # strings. The possible parameters are defined by the individual plugins. For
  # an example, see the description of emTmpConv below.
```

For any modifications of the emFpPlugin files to take effect, Eagle Mode must be restarted.

2.2 emTmpConv - temporary conversion

emTmpConv is a special file viewer which converts the file temporarily to another file format in order to show it through another plugin. The conversion is performed by small shell scripts defined directly in the emFpPlugin configuration.

The library is called `emTmpConv` and the function is `emTmpConvFpPluginFunc`. Three properties are defined, where the third one is optional:

```
Properties = {

  { Name = "OutFileEnding" Value = "insert your ending here" }
    # Specifies the file name ending of the temporary file, including the
    # leading dot. This is important because the final file viewer is chosen by
    # that. For example, if you want the temporary file to be shown by the
    # PostScript viewer, say ".ps".

  { Name="Command" Value="insert your script here" }
    # Specifies the shell script for performing the conversion. It is executed
    # by emTmpConv with /bin/sh -c <script> (on Windows: cmd /E:ON /F:OFF
    # /V:ON /C <script>). Two important environment variables are defined:
    #  $INFILE  (on Windows: %INFILE%)  - Absolute path of the original file to
    #                                     be read by the script.
    #  $OUTFILE (on Windows: %OUTFILE%) - Absolute path of the temporary file
    #                                     (or directory) to be created by the
    #                                     script.
    # On failure, the script should print an error message to stderr and it
    # should exit with a non-zero return value.

  { Name="TunnelFactor" Value="1.0" }
    # This optional property specifies a factor for the depth of the displayed
    # tunnel. The default is 1.0. Zero means to show no tunnel (then it is even
    # possible to enter or leave the content with the auto play functions). A
```

```
        # number less than 1.0 should only be given if the temporary conversion is
        # never expensive in time and (disk) space. With 1.0, the user can trigger
        # only one conversion at a time (per view). The smaller the tunnel factor,
        # the more files can have converted state in parallel.
    }
```

Hints for making the shell scripts:

- All double-quotes and backslashes have to be quoted by backslashes.
- It is always a good idea to resolve path variables with double-quotes, for the case a path contains spaces (and remember to quote the double-quotes). Thus, say `\"$INFILE\"` and `\"$OUTFILE\"` instead of just `$INFILE` and `$OUTFILE` (on Windows please say `\"%INFILE%\"` and `\"%OUTFILE%\"`).
- The script and the programs it calls should be quite stable and secure. No file must be created, modified or deleted falsely.
- The script should not depend on the current working directory. It is undefined per definition.
- On Windows, the script must not contain any line breaks.
- On Windows, a pipe (|) in the script won't work.

For examples of emTmpConv configurations, please see all the files named like `emTmpConv_*.emFpPlugin`.

3 Virtual cosmos configuration

3.1 Cosmos items and cosmos item files

The virtual cosmos of Eagle Mode can still not be edited graphically. But it is possible to edit or create some files manually in order to modify the cosmos.

There are stars and items in the virtual cosmos. The stars cannot be modified, but the items. Practically, each item is a file which is shown (and possibly modified) through an emFpPlugin, just like the files in the file manager. In addition to the shown file, each cosmos item needs some configuration parameters like the position, title and border. Therefore we have two files for each item: a content file and a configuration file.

The configuration file of an item must have the emVcItem file format (described more below) and it must lie in a certain directory named VcItems. The content file of an item can have any type and must lie in a certain directory named VcItemFiles.

Now you may ask where the VcItems and VcItemFiles directories are. The default answer is this:

On UNIX/Linux:

```
$EM_DIR/etc/emMain/VcItems
$EM_DIR/etc/emMain/VcItemFiles
```

On Windows:

```
%EM_DIR%\etcw\emMain\VcItems
%EM_DIR%\etcw\emMain\VcItemFiles
```

($EM_DIR or %EM_DIR% means the installation directory of Eagle Mode)

But those directories are in the host configuration. If you want to edit the cosmos as user configuration, you can copy the directories to the following location and modify them there:

On UNIX/Linux:

```
$HOME/.eaglemode/emMain/VcItems
$HOME/.eaglemode/emMain/VcItemFiles
```

On Windows:

```
%APPDATA%\eaglemode\emMain\VcItems
%APPDATA%\eaglemode\emMain\VcItemFiles
```

3.2 emVcItem file format

As already said, emVcItem files are configuration files for the items of the virtual cosmos. The file name of an emVcItem file must end with `.emVcItem`. The content is a simple ASCII format, and the first line must always be:

```
#%rec:emVirtualCosmosItem%#
```

All further lines are property assignments. Below is a description of all possible properties. You need at least PosX, PosY, Width and FileName.

`Title =` "*title*"
: A title text to be shown in the border of the item.

`PosX =` *number*
`PosY =` *number*
: X and y position of the item within the cosmos. More precise, it is the position of the upper-left corner of the item border. The origin of the cosmos coordinate system is in the upper-left corner of the cosmos. The x axis points to the right, and the y axis points to the bottom. Valid range is 0.0 to 1.0 for both axes.

`Width =` *number*
: Width of the cosmos item including the border.

`ContentTallness =` *number*
: Height/width ratio of the cosmos item excluding the border (e.g. say 0.5625 if you want an aspect ratio of 16:9).

`BorderScaling =` *number*
: Scale factor for the size of the border. Say 0.0 if you don't want a border.

`BackgroundColor =` *color*
`BorderColor =` *color*
`TitleColor =` *color*

Colors of background, border and title text. A color can be specified as a color name like "Powder Blue", or as a hexadecimal RGB value string like "#B0E0E6" or "#BEE". Another possibility is to give red, green, blue and alpha components as decimal bytes in curly braces (e.g. TitleColor = { 255 0 0 128 }).

Focusable = *boolean*
: Whether the content panel shall be focusable. Possible values are: true (the default) and false.

FileName = "*file name*"
: Name of the cosmos item file. It must be the name of a file in the VcItemFiles directory. If you want to specify a path, please do not specify it here, and create an emFileLink file in the VcItemFiles directory instead.

CopyToUser = *boolean*
: Whether the cosmos item file shall be copied to user configuration and be used there. This is useful for all cosmos item files where the plugin requires write access because it is more than a viewer. Possible values are: true and false (the default).

Alternative = *integer number*
: Index to the alternative file panel plugins. The default of 0 means to use the original plugin. For example, if you want the hex viewer instead of plain text, say 1.

3.3 emFileLink file format

emFileLink is a plugin application that defines a file format for links to files and directories. It is mainly used to show external files and directories in the virtual cosmos without copying them into the cosmos files directory. The file format is also called emFileLink. An emFileLink file simply shows the contents of the referred file or directory with a suitable emFpPlugin, but optionally with a directory entry panel around it.

The file names of emFileLink files must end with .emFileLink.

Now to the file format. It is a simple ASCII format. An example for an absolute link to /usr/share is:

```
#%rec:emFileLink%#
Path="/usr/share"
```

If you want a directory entry panel to be shown, add this line:

```
HaveDirEntry=yes
```

Furthermore, it is possible to define the path relative to a generic base path. For example, if you want a link to $HOME/pictures, you could say:

```
#%rec:emFileLink%#
BasePathType=Home
Path="pictures"
```

Possible values for BasePathType are: None (the default), Bin, Include, Lib, HtmlDoc, PdfDoc, PsDoc, UserConfig, HostConfig, Tmp, Res, Home. Some of these also require a project name which can be given with the BasePathProject property. Here comes an example which links to the bookmarks file of the emMain project:

```
#%rec:emFileLink%#
BasePathType=UserConfig
BasePathProject="emMain"
Path="bookmarks.rec"
HaveDirEntry=yes
```