frei0r Reference Manual

Generated by Doxygen 1.5.1

Thu May 31 17:38:09 2007

Contents

Chapter 1

frei0r - a minimalistic plugin API for video effects

1.1 Introduction

This is frei0r - a minimalistic plugin API for video effects.

The main emphasis is on simplicity - there are many different applications that use video effects, and they all have different requirements regarding their internal plugin API. And that's why frei0r does not try to be a one-in-all general video plugin API, but instead an API for the most common video effects: simple filters, sources and mixers that can be controlled by parameters.

It's our hope that this way these simple effects can be shared between many applications, avoiding their reimplementation by different projects.

On the other hand, this is not meant as a competing standard to more ambitious efforts that try to satisfy the needs of many different applications and more complex effects.

1.2 Overview

If you are new to frei0r, the best thing is probably to have a look at the `frei0r header`, which is quite simple.

After that, you might want to look at the `frei0r functions` in more detail.

When developing a new frei0r effect, you have to choose

- which effect type to use (**Type of the Plugin** (p. 12)),
- which color model to use (**Color Models** (p. 13)), and
- which parameter types (**Parameter Types** (p. 15)) your effect will support.

To round things up, you should decide whether your effect should have an associated icon (**Icons for frei0r effects** (p. 10)), and where it will be installed (**Plugin Locations** (p. 9)).

1.3 Changes

1.3.1 From frei0r 1.0 to frei0r 1.1

- added specifications for plugin locations
- added specifications for frei0r icons
- added RGBA8888 color model
- added packed32 color model
- added better specification of color models
- added string type
- added bounds to resolution (8 <= width, height <= 2048)
- width and height must be an integer multiple of 8
- frame data must be 16 byte aligned
- improved update specification (must not change parameters, must restore fpu state)
- added note for applications to ignore effects with unknown fields
- added new plugin types mixer2 and mixer3
- added section about **Concurrency** (p. 11)

Chapter 2

frei0r Module Index

2.1 frei0r Modules

Here is a list of all modules:

Chapter 3

frei0r Data Structure Index

3.1 frei0r Data Structures

Here are the data structures with brief descriptions:

Chapter 4

frei0r File Index

4.1 frei0r File List

Here is a list of all documented files with brief descriptions:

Chapter 5

frei0r Module Documentation

5.1 Plugin Locations

5.1.1 Plugin Locations

For Unix platforms there are rules for the location of frei0r plugins.

frei0r 1.x plugin files should be located in

- (1) /usr/lib/frei0r-1/<vendor>
- (2) /usr/local/lib/frei0r-1/<vendor>
- (3) $HOME/.frei0r-1/lib/<vendor>

Examples:

- /usr/lib/frei0r-1/mob/flippo.so
- /usr/lib/frei0r-1/drone/flippo.so
- /usr/local/lib/frei0r-1/gephex/coma/invert0r.so
- /home/martin/.frei0r-1/lib/martin/test.so

Like in these examples plugins should be placed in "vendor" subdirs to reduce name clashes.

5.1.1.1 Plugin Loading Order

The application shall load plugins in the following order: 3, 2, 1. If a name clash occurs (two or more frei0r plugins with identical effect name), the plugins in directory 3 have precedence over plugins in directory 2, and those in directory 2 have precedence over plugins in directory 1.

This makes it possible for users to "override" effects that are installed in system wide directories by placing plugins in their home directory.

The order of loading plugins inside each of the directories 1, 2, and 3 is not defined.

5.2 Icons for frei0r effects

5.2.1 Icons for frei0r effects

Each frei0r effect can have an associated icon.

5.2.1.1 Icon Format

The format of frei0r icons must be png. Recommended resolution is 64x64. The icon filename of an effect with effect name "frei0r" must be "frei0r.png".

5.2.1.2 Icon location

The exact location where the application should look for the plugin is platform dependant.

For Windows platforms, the icon should be at the same place as the plugin containing the effect.

For Unix platforms, the following mapping from plugin location to icon location must be used:

Let <plugin_path>/<plugin> be a frei0r plugin with name <effect_name>. Then the corresponding icon (if any) shall be located in <icon_path>/<effect_name>.png. <icon_path> can be obtained in the following way:

<plugin_path>	<icon_path>
$HOME/.frei0r-1/lib/<vendor>	$HOME/.frei0r-1/icons/<vendor>
/usr/local/lib/frei0r-1/<vendor>	/usr/local/share/frei0r-1/icons/<vendor>
/usr/lib/frei0r-1/<vendor>	/usr/share/frei0r-1/icons/<vendor>
*	<plugin_path>

(The wildcard '*' stands for any other plugin_path)

For other platforms, no location is defined. We recommend to use the plugin path where possible.

5.3 Concurrency

5.3.1 Concurrency

- **f0r_init** (p. 26)
- **f0r_deinit** (p. 25)

These methods must not be called more than once. It is obvious that no concurrent calls are allowed.

- **f0r_get_plugin_info** (p. 26)
- **f0r_get_param_info** (p. 26)
- **f0r_construct** (p. 25)
- **f0r_destruct** (p. 26)

Concurrent calls of these functions are allowed.

- **f0r_set_param_value** (p. 27)
- **f0r_get_param_value** (p. 26)
- **f0r_update** (p. 27)
- **f0r_update2** (p. 27)

If a thread is in one of these methods its allowed for another thread to enter one of theses methods for a different effect instance. But for one effect instance only one thread is allowed to execute any of these methods.

5.4 Type of the Plugin

Defines

- #define **F0R_PLUGIN_TYPE_FILTER** 0
- #define **F0R_PLUGIN_TYPE_SOURCE** 1
- #define **F0R_PLUGIN_TYPE_MIXER2** 2
- #define **F0R_PLUGIN_TYPE_MIXER3** 3

5.4.1 Detailed Description

These defines determine whether the plugin is a source, a filter or one of the two mixer types

5.4.2 Define Documentation

5.4.2.1 #define F0R_PLUGIN_TYPE_FILTER 0

one input and one output

5.4.2.2 #define F0R_PLUGIN_TYPE_MIXER2 2

two inputs and one output

5.4.2.3 #define F0R_PLUGIN_TYPE_MIXER3 3

three inputs and one output

5.4.2.4 #define F0R_PLUGIN_TYPE_SOURCE 1

just one output

5.5 Color Models

Defines

- #define **F0R_COLOR_MODEL_BGRA8888** 0
- #define **F0R_COLOR_MODEL_RGBA8888** 1
- #define **F0R_COLOR_MODEL_PACKED32** 2

5.5.1 Detailed Description

List of supported color models.

Note: the color models are endian independent, because the color components are defined by their positon in memory, not by their significance in an uint32_t value.

For effects that work on the color components, RGBA8888 is the recommended color model for frei0r-1.1 effects. For effects that only work on pixels, PACKED32 is the recommended color model since it helps the application to avoid unnecessary color conversions.

Effects can choose an appropriate color model, applications must support all color models and do conversions if necessary. Source effects must not use the PACKED32 color model because the application must know in which color model the created framebuffers are represented.

For each color model, a frame consists of width∗height pixels which are stored row-wise and consecutively in memory. The size of a pixel is 4 bytes. There is no extra pitch parameter (i.e. the pitch is simply width∗4).

The following additional constraints must be honored:

- The top-most line of a frame is stored first in memory.
- A frame must be aligned to a 16 byte border in memory.
- The width and height of a frame must be positive
- The width and height of a frame must be integer multiples of 8

These constraints make sure that each line is stored at an address aligned to 16 byte.

5.5.2 Define Documentation

5.5.2.1 #define F0R_COLOR_MODEL_BGRA8888 0

In BGRA8888, each pixel is represented by 4 consecutive unsigned bytes, where the first byte value represents the blue, the second the green, and the third the red color component of the pixel. The last value represents the alpha value.

5.5.2.2 #define F0R_COLOR_MODEL_PACKED32 2

In PACKED32, each pixel is represented by 4 consecutive bytes, but it is not defined how the color componets are stored. The true color format could be RGBA8888, BGRA8888, a packed 32 bit YUV format, or any other color format that stores pixels in 32 bit.

This is useful for effects that don't work on color but only on pixels (for example a mirror effect).

Note that source effects must not use this color model.

5.5.2.3 #define F0R_COLOR_MODEL_RGBA8888 1

In RGBA8888, each pixel is represented by 4 consecutive unsigned bytes, where the first byte value represents the red, the second the green, and the third the blue color component of the pixel. The last value represents the alpha value.

5.6 Parameter Types

Data Structures

- struct **f0r_param_color**
- struct **f0r_param_position**

Defines

- #define **F0R_PARAM_BOOL** 0
- #define **F0R_PARAM_DOUBLE** 1
- #define **F0R_PARAM_COLOR** 2
- #define **F0R_PARAM_POSITION** 3
- #define **F0R_PARAM_STRING** 4

Typedefs

- typedef double **f0r_param_bool**
- typedef double **f0r_param_double**
- typedef **f0r_param_color** **f0r_param_color_t**
- typedef **f0r_param_position** **f0r_param_position_t**
- typedef char **f0r_param_string**

5.6.1 Define Documentation

5.6.1.1 #define F0R_PARAM_BOOL 0

Parameter type for boolean values

See also:

f0r_param_bool (p. 16)

5.6.1.2 #define F0R_PARAM_COLOR 2

Parameter type for color

See also:

f0r_param_color (p. 17)

5.6.1.3 #define F0R_PARAM_DOUBLE 1

Parameter type for doubles

See also:

f0r_param_double (p. 16)

5.6.1.4 #define F0R_PARAM_POSITION 3

Parameter type for position

See also:

f0r_param_position (p. 19)

5.6.1.5 #define F0R_PARAM_STRING 4

Parameter type for string

See also:

f0r_param_string (p. 16)

5.6.2 Typedef Documentation

5.6.2.1 typedef double f0r_param_bool

The boolean type. The allowed range of values is [0, 1]. [0, 0.5[is mapped to false and [0.5, 1] is mapped to true.

5.6.2.2 typedef struct f0r_param_color f0r_param_color_t

The color type. All three color components are in the range [0, 1].

5.6.2.3 typedef double f0r_param_double

The double type. The allowed range of values is [0, 1].

5.6.2.4 typedef struct f0r_param_position f0r_param_position_t

The position type. Both position coordinates are in the range [0, 1].

5.6.2.5 typedef char f0r_param_string

The string type. Zero terminated array of 8-bit values in utf-8 encoding

Chapter 6

frei0r Data Structure Documentation

6.1 f0r_param_color Struct Reference

`#include <frei0r.h>`

Data Fields

- float **r**
- float **g**
- float **b**

6.1.1 Detailed Description

The color type. All three color components are in the range [0, 1].

6.1.2 Field Documentation

6.1.2.1 float f0r_param_color::r

red color component

6.1.2.2 float f0r_param_color::g

green color component

6.1.2.3 float f0r_param_color::b

blue color component

The documentation for this struct was generated from the following file:

- **frei0r.h**

6.2 f0r_param_info Struct Reference

`#include <frei0r.h>`

Data Fields

- const char * **name**
- int **type**
- const char * **explanation**

6.2.1 Detailed Description

Similar to f0r_plugin_info_t, this structure is filled by the plugin for every parameter.

All strings are unicode, 0-terminated, and the encoding is utf-8.

6.2.2 Field Documentation

6.2.2.1 const char* f0r_param_info::name

The (short) name of the param

6.2.2.2 int f0r_param_info::type

The type (see the F0R_PARAM_* defines)

6.2.2.3 const char* f0r_param_info::explanation

Optional explanation (can be 0)

The documentation for this struct was generated from the following file:

- **frei0r.h**

6.3 f0r_param_position Struct Reference

`#include <frei0r.h>`

Data Fields

- double **x**
- double **y**

6.3.1 Detailed Description

The position type. Both position coordinates are in the range [0, 1].

6.3.2 Field Documentation

6.3.2.1 double f0r_param_position::x

x coordinate

6.3.2.2 double f0r_param_position::y

y coordinate

The documentation for this struct was generated from the following file:

- **frei0r.h**

6.4 f0r_plugin_info Struct Reference

`#include <frei0r.h>`

Data Fields

- const char * **name**
- const char * **author**
- int **plugin_type**
- int **color_model**
- int **frei0r_version**
- int **major_version**
- int **minor_version**
- int **num_params**
- const char * **explanation**

6.4.1 Detailed Description

The f0r_plugin_info_t structure is filled in by the plugin to tell the application about its name, type, number of parameters, and version.

An application should ignore (i.e. not use) frei0r effects that have unknown values in the plugin_type or color_model field. It should also ignore effects with a too high frei0r_version.

This is necessary to be able to extend the frei0r spec (e.g. by adding new color models or plugin types) in a way that does not result in crashes when loading effects that make use of these extensions into an older application.

All strings are unicode, 0-terminated, and the encoding is utf-8.

6.4.2 Field Documentation

6.4.2.1 const char* f0r_plugin_info::name

The (short) name of the plugin

6.4.2.2 const char* f0r_plugin_info::author

The plugin author

6.4.2.3 int f0r_plugin_info::plugin_type

The plugin type

See also:

Type of the Plugin (p. 12)

6.4.2.4 int f0r_plugin_info::color_model

The color model used

6.4.2.5 int f0r_plugin_info::frei0r_version

The frei0r major version this plugin is built for

6.4.2.6 int f0r_plugin_info::major_version

The major version of the plugin

6.4.2.7 int f0r_plugin_info::minor_version

The minor version of the plugin

6.4.2.8 int f0r_plugin_info::num_params

The number of parameters of the plugin

6.4.2.9 const char* f0r_plugin_info::explanation

An optional explanation string

The documentation for this struct was generated from the following file:

- **frei0r.h**

Chapter 7

frei0r File Documentation

7.1 frei0r.h File Reference

This file defines the frei0r api, version 1.1.

```
#include <inttypes.h>
```

Data Structures

- struct **f0r_plugin_info**
- struct **f0r_param_color**
- struct **f0r_param_position**
- struct **f0r_param_info**

Defines

- #define **FREI0R_MAJOR_VERSION** 1
- #define **FREI0R_MINOR_VERSION** 1
- #define **F0R_PLUGIN_TYPE_FILTER** 0
- #define **F0R_PLUGIN_TYPE_SOURCE** 1
- #define **F0R_PLUGIN_TYPE_MIXER2** 2
- #define **F0R_PLUGIN_TYPE_MIXER3** 3
- #define **F0R_COLOR_MODEL_BGRA8888** 0
- #define **F0R_COLOR_MODEL_RGBA8888** 1
- #define **F0R_COLOR_MODEL_PACKED32** 2
- #define **F0R_PARAM_BOOL** 0
- #define **F0R_PARAM_DOUBLE** 1
- #define **F0R_PARAM_COLOR** 2
- #define **F0R_PARAM_POSITION** 3
- #define **F0R_PARAM_STRING** 4

Typedefs

- typedef **f0r_plugin_info f0r_plugin_info_t**
- typedef double **f0r_param_bool**

- typedef double **f0r_param_double**
- typedef **f0r_param_color** **f0r_param_color_t**
- typedef **f0r_param_position** **f0r_param_position_t**
- typedef char **f0r_param_string**
- typedef **f0r_param_info** **f0r_param_info_t**
- typedef void * **f0r_instance_t**
- typedef void * **f0r_param_t**

Functions

- int **f0r_init** ()
- void **f0r_deinit** ()
- void **f0r_get_plugin_info** (**f0r_plugin_info_t** *info)
- void **f0r_get_param_info** (**f0r_param_info_t** *info, int param_index)
- **f0r_instance_t** **f0r_construct** (unsigned int width, unsigned int height)
- void **f0r_destruct** (**f0r_instance_t** instance)
- void **f0r_set_param_value** (**f0r_instance_t** instance, **f0r_param_t** param, int param_index)
- void **f0r_get_param_value** (**f0r_instance_t** instance, **f0r_param_t** param, int param_index)
- void **f0r_update** (**f0r_instance_t** instance, double time, const uint32_t *inframe, uint32_t *outframe)
- void **f0r_update2** (**f0r_instance_t** instance, double time, const uint32_t *inframe1, const uint32_t *inframe2, const uint32_t *inframe3, uint32_t *outframe)

7.1.1 Detailed Description

This file defines the frei0r api, version 1.1.

A conforming plugin must implement and export all functions declared in this header.

A conforming application must accept only those plugins which use allowed values for the described fields.

7.1.2 Define Documentation

7.1.2.1 #define FREI0R_MAJOR_VERSION 1

The frei0r API major version

7.1.2.2 #define FREI0R_MINOR_VERSION 1

The frei0r API minor version

7.1.3 Typedef Documentation

7.1.3.1 typedef void* f0r_instance_t

Transparent instance pointer of the frei0r effect.

7.1.3.2 typedef struct f0r_param_info f0r_param_info_t

Similar to f0r_plugin_info_t, this structure is filled by the plugin for every parameter.

All strings are unicode, 0-terminated, and the encoding is utf-8.

7.1.3.3 typedef void* f0r_param_t

Transparent parameter handle.

7.1.3.4 typedef struct f0r_plugin_info f0r_plugin_info_t

The f0r_plugin_info_t structure is filled in by the plugin to tell the application about its name, type, number of parameters, and version.

An application should ignore (i.e. not use) frei0r effects that have unknown values in the plugin_type or color_model field. It should also ignore effects with a too high frei0r_version.

This is necessary to be able to extend the frei0r spec (e.g. by adding new color models or plugin types) in a way that does not result in crashes when loading effects that make use of these extensions into an older application.

All strings are unicode, 0-terminated, and the encoding is utf-8.

7.1.4 Function Documentation

7.1.4.1 f0r_instance_t f0r_construct (unsigned int *width*, unsigned int *height*)

Constructor for effect instances. The plugin returns a pointer to its internal instance structure.

The resolution has to be an integer multiple of 8, must be greater than 0 and be at most 2048 in both dimensions.

Parameters:

width The x-resolution of the processed video frames

height The y-resolution of the processed video frames

Returns:

0 on failure or a pointer != 0 on success

See also:

f0r_destruct (p. 26)

7.1.4.2 void f0r_deinit ()

f0r_deinit is called once when the plugin is unloaded by the application.

See also:

f0r_init (p. 26)

7.1.4.3 void f0r_destruct (f0r_instance_t *instance*)

Destroys an effect instance.

Parameters:

instance The pointer to the plugins internal instance structure.

See also:

f0r_construct (p. 25)

7.1.4.4 void f0r_get_param_info (f0r_param_info_t * *info*, int *param_index*)

f0r_get_param_info is called by the application to query the type of each parameter.

Parameters:

info is allocated by the application and filled by the plugin

param_index the index of the parameter to be queried (from 0 to num_params-1)

7.1.4.5 void f0r_get_param_value (f0r_instance_t *instance*, f0r_param_t *param*, int *param_index*)

This function allows the application to query the parameter values of an effect instance.

Parameters:

instance the effect instance

param pointer to the parameter value

param_index index of the parameter

See also:

f0r_set_param_value (p. 27)

7.1.4.6 void f0r_get_plugin_info (f0r_plugin_info_t * *info*)

Is called once after init. The plugin has to fill in the values in info.

Parameters:

info Pointer to an info struct allocated by the application.

7.1.4.7 int f0r_init ()

f0r_init() (p. 26) is called once when the plugin is loaded by the application.

See also:

f0r_deinit (p. 25)

7.1.4.8 void f0r_set_param_value (f0r_instance_t *instance*, f0r_param_t *param*, int *param_index*)

This function allows the application to set the parameter values of an effect instance. Validity of the parameter pointer is handled by the application thus the data must be copied by the effect.

Parameters:

instance the effect instance

param pointer to the parameter value

param_index index of the parameter

See also:

f0r_get_param_value (p. 26)

7.1.4.9 void f0r_update (f0r_instance_t *instance*, double *time*, const uint32_t * *inframe*, uint32_t * *outframe*)

This is where the core effect processing happens. The application calls it after it has set the necessary parameter values. inframe and outframe must be aligned to an integer multiple of 16 bytes in memory.

This funcition should not alter the parameters of the effect in any way (**f0r_get_param_value** (p. 26) should return the same values after a call to **f0r_update** (p. 27) as before the call).

The function is responsible to restore the fpu state (e.g. rounding mode) and mmx state if applicable before it returns to the caller.

The host mustn't call **f0r_update** (p. 27) for effects of type **F0R_PLUGIN_TYPE_MIXER2** (p. 12) and **F0R_PLUGIN_TYPE_MIXER3** (p. 12).

Parameters:

instance the effect instance

time the application time in seconds but with subsecond resolution (e.g. milli-second resolution). The resolution should be at least the inter-frame period of the application.

inframe the incoming video frame (can be zero for sources)

outframe the resulting video frame

See also:

f0r_update2 (p. 27)

7.1.4.10 void f0r_update2 (f0r_instance_t *instance*, double *time*, const uint32_t * *inframe1*, const uint32_t * *inframe2*, const uint32_t * *inframe3*, uint32_t * *outframe*)

For effects of type **F0R_PLUGIN_TYPE_SOURCE** (p. 12) or **F0R_PLUGIN_TYPE_FILTER** (p. 12) this method is optional. The **f0r_update** (p. 27) method must still be exported for these two effect types. If both are provided the behavior of them must be the same.

Effects of type **F0R_PLUGIN_TYPE_MIXER2** (p. 12) or **F0R_PLUGIN_TYPE_MIXER3** (p. 12) must provide the new **f0r_update2** (p. 27) method.

Parameters:

instance the effect instance

time the application time in seconds but with subsecond resolution (e.g. milli-second resolution). The resolution should be at least the inter-frame period of the application.

inframe1 the first incoming video frame (can be zero for sources)

inframe2 the second incoming video frame (can be zero for sources and filters)

inframe3 the third incoming video frame (can be zero for sources, filters and mixer3)

outframe the resulting video frame

See also:

f0r_update (p. 27)

Index