# Ecore Reference Manual

Generated by Doxygen 1.5.1

Wed Mar 28 00:01:11 2007

# Contents

# Chapter 1

# Ecore

Figure 1.1: width=5cm

**Version:**

1.0.0

**Author:**

Carsten Haitzler <raster@rasterman.com>
Tom Gilbert <tom@linuxbrit.co.uk>
Burra <burra@colorado.edu>
Chris Ross <chris@darkrock.co.uk>
Term <term@twistedpath.org>
Tilman Sauerbeck <tilman@code-monkey.de>
Nathan Ingersoll <rbdpngn@users.sourceforge.net>

**Date:**

2000-2004

## 1.1 What is Ecore?

Ecore is a library of convenience functions.

The Ecore library provides the following modules:

- Ecore - Main Loop Functions.
- Ecore_Con - Connection functions.
- Ecore_Config - Configuration functions.
- Ecore_Evas - Evas convenience functions.
- Ecore_FB - Frame buffer convenience functions.
- Ecore_IPC - Inter Process Communication functions.
- Ecore_Job - Job functions, to be used in the Ecore main loop.
- Ecore_Txt - Text encoding conversion.
- Ecore_X - X Windows System wrapper.

## 1.2 How to compile using Ecore?

This section has to be documented. Below is just a quick line to handle all Ecore modules at once.

```
gcc *.c \
-I/usr/local/include -I/usr/X11R6/include \
-L/usr/local/lib -L/usr/X11R6/lib \
-lecore -lecore_evas -lecore_x -lecore_fb -lecore_job \
`evas-config --cflags --libs`
```

## 1.3 How is it installed?

Suggested configure options for evas for a Linux desktop X display:

```
./configure \
--enable-ecore-x \
--enable-ecore-fb \
--enable-ecore-evas \
--enable-ecore-evas-gl \
--enable-ecore-job \
--enable-ecore-con \
--enable-ecore-ipc \
--enable-ecore-txt
make CFLAGS="-O9 -mpentiumpro -march=pentiumpro -mcpu=pentiumpro"
```

**Todo**

(1.0) Document API

*/

/*

# Chapter 2

# Ecore Module Index

## 2.1 Ecore Modules

Here is a list of all modules:

# Chapter 3

# Ecore Data Structure Index

## 3.1 Ecore Data Structures

Here are the data structures with brief descriptions:

# Chapter 4

# Ecore File Index

## 4.1 Ecore File List

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

# Chapter 5

# Ecore Page Index

## 5.1 Ecore Related Pages

Here is a list of all related documentation pages:

# Chapter 6

# Ecore Module Documentation

## 6.1 Ecore Timer

The timer allows callbacks to be called at specific intervals.

## 6.2 Ecore Jobs

You can queue jobs that are to be done by the main loop when the current event is dealt with.

### Functions

- EAPI Ecore_Job * ecore_job_add (void(*func)(void *data), const void *data)
  *Add a job to the event queue.*
- EAPI void * ecore_job_del (Ecore_Job *job)
  *Delete a queued job that has not yet been executed.*

### 6.2.1 Detailed Description

You can queue jobs that are to be done by the main loop when the current event is dealt with.

### 6.2.2 Function Documentation

#### 6.2.2.1 EAPI Ecore_Job* ecore_job_add (void(*)(void *data) *func*, const void * *data*)

Add a job to the event queue.

**Parameters:**

***func*** The function to call when the job gets handled.

***data*** Data pointer to be passed to the job function when the job is handled.

**Returns:**

The handle of the job. `NULL` is returned if the job could not be added to the queue.

**Note:**

Once the job has been executed, the job handle is invalid.

#### 6.2.2.2 EAPI void* ecore_job_del (Ecore_Job * *job*)

Delete a queued job that has not yet been executed.

**Parameters:**

***job*** Handle of the job to delete.

**Returns:**

The data pointer that was to be passed to the job.

## 6.3 Idle Handlers

Callbacks that are called when the program enters or exits an idle state.

### Functions

- EAPI Ecore_Idle_Enterer * ecore_idle_enterer_add (int(*func)(void *data), const void *data)

  *Add an idle enterer handler.*

- EAPI void * ecore_idle_enterer_del (Ecore_Idle_Enterer *idle_enterer)

  *Delete an idle enterer callback.*

- EAPI Ecore_Idle_Exiter * ecore_idle_exiter_add (int(*func)(void *data), const void *data)

  *Add an idle exiter handler.*

- EAPI void * ecore_idle_exiter_del (Ecore_Idle_Exiter *idle_exiter)

  *Delete an idle exiter handler from the list to be run on exiting idle state.*

- EAPI Ecore_Idler * ecore_idler_add (int(*func)(void *data), const void *data)

  *Add an idler handler.*

- EAPI void * ecore_idler_del (Ecore_Idler *idler)

  *Delete an idler callback from the list to be executed.*

### 6.3.1 Detailed Description

Callbacks that are called when the program enters or exits an idle state.

The ecore main loop enters an idle state when it is waiting for timers to time out, data to come in on a file descriptor or any other event to occur. You can set callbacks to be called when the main loop enters an idle state, during an idle state or just after the program wakes up.

Enterer callbacks are good for updating your program's state, if it has a state engine. Once all of the enterer handlers are called, the program will enter a "sleeping" state.

Idler callbacks are called when the main loop has called all enterer handlers. They are useful for interfaces that require polling and timers would be too slow to use.

If no idler callbacks are specified, then the process literally goes to sleep. Otherwise, the idler callbacks are called continuously while the loop is "idle", using as much CPU as is available to the process.

Exiter callbacks are called when the main loop wakes up from an idle state.

### 6.3.2 Function Documentation

#### 6.3.2.1 EAPI Ecore_Idle_Enterer* ecore_idle_enterer_add (int(*)(void *data) *func*, const void * *data*)

Add an idle enterer handler.

**Parameters:**

***func*** The function to call when entering an idle state.

***data*** The data to be passed to the `func` call

**Returns:**

A handle to the idle enterer callback if successful. Otherwise, NULL is returned.

#### 6.3.2.2 EAPI void* ecore_idle_enterer_del (Ecore_Idle_Enterer * *idle_enterer*)

Delete an idle enterer callback.

**Parameters:**

***idle_enterer*** The idle enterer to delete

**Returns:**

The data pointer passed to the idler enterer callback on success. NULL otherwise.

#### 6.3.2.3 EAPI Ecore_Idle_Exiter* ecore_idle_exiter_add (int(*)(void *data) *func*, const void * *data*)

Add an idle exiter handler.

**Parameters:**

***func*** The function to call when exiting an idle state.

***data*** The data to be passed to the `func` call

**Returns:**

A handle to the idle exiter callback on success. NULL otherwise.

#### 6.3.2.4 EAPI void* ecore_idle_exiter_del (Ecore_Idle_Exiter * *idle_exiter*)

Delete an idle exiter handler from the list to be run on exiting idle state.

**Parameters:**

***idle_exiter*** The idle exiter to delete

**Returns:**

The data pointer that was being being passed to the handler if successful. NULL otherwise.

#### 6.3.2.5 EAPI Ecore_Idler* ecore_idler_add (int(*)(void *data) *func*, const void * *data*)

Add an idler handler.

**Parameters:**

***func*** The function to call when idling.

***data*** The data to be passed to this `func` call.

**Returns:**

A idler handle if successfully added. NULL otherwise.

Add an idler handle to the event loop, returning a handle on success and NULL otherwise. The function `func` will be called repeatedly while no other events are ready to be processed, as long as it returns 1. A return of 0 deletes the idler.

Idlers are useful for progressively prossessing data without blocking.

#### 6.3.2.6 EAPI void* ecore_idler_del (Ecore_Idler * *idler*)

Delete an idler callback from the list to be executed.

**Parameters:**

***idler*** The handle of the idler callback to delete

**Returns:**

The data pointer passed to the idler callback on success. NULL otherwise.

## 6.4 Ecore Config Create Functions

Convenience functions that set default values, bounds, option values and descriptions in one call.

### Functions

- int ecore_config_create (const char *key, void *val, char short_opt, char *long_opt, char *desc)

  *Creates a new property, if it does not already exist, and sets its attributes to those given.*

- int ecore_config_typed_create (const char *key, void *val, int type, char short_opt, char *long_opt, char *desc)

  *Creates a new property, if it does not already exist, and sets its attributes to those given.*

- int ecore_config_boolean_create (const char *key, int val, char short_opt, char *long_opt, char *desc)

  *Creates a new boolean property, if it does not already exist, and sets its attributes to those given.*

- int ecore_config_int_create (const char *key, int val, char short_opt, char *long_opt, char *desc)

  *Creates a new integer property, if it does not already exist, and sets its attributes to those given.*

- int ecore_config_int_create_bound (const char *key, int val, int low, int high, int step, char short_opt, char *long_opt, char *desc)

  *Creates a new integer property, if it does not already exist, and sets its attributes to those given.*

- int ecore_config_string_create (const char *key, char *val, char short_opt, char *long_opt, char *desc)

  *Creates a new string property, if it does not already exist, and sets its attributes to those given.*

- int ecore_config_float_create (const char *key, float val, char short_opt, char *long_opt, char *desc)

  *Creates a new float property, if it does not already exist, and sets its attributes to those given.*

- int ecore_config_float_create_bound (const char *key, float val, float low, float high, float step, char short_opt, char *long_opt, char *desc)

  *Creates a new float property, if it does not already exist, and sets its attributes to those given.*

- int ecore_config_argb_create (const char *key, char *val, char short_opt, char *long_opt, char *desc)

  *Creates a new color property, if it does not already exist, and sets its attributes to those given.*

- int ecore_config_theme_create (const char *key, char *val, char short_opt, char *long_opt, char *desc)

  *Creates a new theme property, if it does not already exist, and sets its attributes to those given.*

### 6.4.1 Detailed Description

Convenience functions that set default values, bounds, option values and descriptions in one call.

### 6.4.2 Function Documentation

#### 6.4.2.1 int ecore_config_argb_create (const char * *key*, char * *val*, char *short_opt*, char * *long_opt*, char * *desc*)

Creates a new color property, if it does not already exist, and sets its attributes to those given.

**Parameters:**

***key*** The property key.

***val*** Default color value of key, as a hexadecimal string.

***short_opt*** Short option used to set the property from command line.

***long_opt*** Long option used to set the property from command line.

***desc*** String description of property.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` on success.

#### 6.4.2.2 int ecore_config_boolean_create (const char * *key*, int *val*, char *short_opt*, char * *long_opt*, char * *desc*)

Creates a new boolean property, if it does not already exist, and sets its attributes to those given.

**Parameters:**

***key*** The property key.

***val*** Default boolean value of key.

***short_opt*** Short option used to set the property from command line.

***long_opt*** Long option used to set the property from command line.

***desc*** String description of property.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` on success.

#### 6.4.2.3 int ecore_config_create (const char * *key*, void * *val*, char *short_opt*, char * *long_opt*, char * *desc*)

Creates a new property, if it does not already exist, and sets its attributes to those given.

The type of the property is guessed from the key and the value given.

**Parameters:**

***key*** The property key.

***val*** Pointer to default value of key.

***short_opt*** Short option used to set the property from command line.

***long_opt*** Long option used to set the property from command line.

***desc*** String description of property.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` on success.

#### 6.4.2.4 int ecore_config_float_create (const char * *key*, float *val*, char *short_opt*, char * *long_opt*, char * *desc*)

Creates a new float property, if it does not already exist, and sets its attributes to those given.

**Parameters:**

***key*** The property key.

***val*** Default float value of key.

***short_opt*** Short option used to set the property from command line.

***long_opt*** Long option used to set the property from command line.

***desc*** String description of property.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` on success.

#### 6.4.2.5 int ecore_config_float_create_bound (const char * *key*, float *val*, float *low*, float *high*, float *step*, char *short_opt*, char * *long_opt*, char * *desc*)

Creates a new float property, if it does not already exist, and sets its attributes to those given.

**Parameters:**

***key*** The property key.

***val*** Default float value of key.

***low*** Lowest valid float value for the property.

***high*** Highest valid float value for the property.

***step*** Increment value for the property.

***short_opt*** Short option used to set the property from command line.

***long_opt*** Long option used to set the property from command line.

***desc*** String description of property.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` on success.

#### 6.4.2.6 int ecore_config_int_create (const char * *key*, int *val*, char *short_opt*, char * *long_opt*, char * *desc*)

Creates a new integer property, if it does not already exist, and sets its attributes to those given.

**Parameters:**

***key*** The property key.

***val*** Default integer value of key.

***short_opt*** Short option used to set the property from command line.

***long_opt*** Long option used to set the property from command line.

***desc*** String description of property.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` on success.

#### 6.4.2.7 int ecore_config_int_create_bound (const char * *key*, int *val*, int *low*, int *high*, int *step*, char *short_opt*, char * *long_opt*, char * *desc*)

Creates a new integer property, if it does not already exist, and sets its attributes to those given.

**Parameters:**

***key*** The property key.

***val*** Default integer value of key.

***low*** Lowest valid integer value for the property.

***high*** Highest valid integer value for the property.

***step*** Increment value for the property.

***short_opt*** Short option used to set the property from command line.

***long_opt*** Long option used to set the property from command line.

***desc*** String description of property.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` on success.

#### 6.4.2.8 int ecore_config_string_create (const char * *key*, char * *val*, char *short_opt*, char * *long_opt*, char * *desc*)

Creates a new string property, if it does not already exist, and sets its attributes to those given.

**Parameters:**

***key*** The property key.
***val*** Default value of key.
***short_opt*** Short option used to set the property from command line.
***long_opt*** Long option used to set the property from command line.
***desc*** String description of property.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` on success.

#### 6.4.2.9 int ecore_config_theme_create (const char * *key*, char * *val*, char *short_opt*, char * *long_opt*, char * *desc*)

Creates a new theme property, if it does not already exist, and sets its attributes to those given.

**Parameters:**

***key*** The property key.
***val*** Default theme name for the property.
***short_opt*** Short option used to set the property from command line.
***long_opt*** Long option used to set the property from command line.
***desc*** String description of property.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` on success.

#### 6.4.2.10 int ecore_config_typed_create (const char * *key*, void * *val*, int *type*, char *short_opt*, char * *long_opt*, char * *desc*)

Creates a new property, if it does not already exist, and sets its attributes to those given.

**Parameters:**

***key*** The property key.
***val*** Pointer to default value of key.
***type*** Type of the property.
***short_opt*** Short option used to set the property from command line.
***long_opt*** Long option used to set the property from command line.
***desc*** String description of property.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` on success.

## 6.5 Ecore Config File Functions

Functions that are used to load and save properties from and to files.

### Functions

- EAPI int ecore_config_load (void)
  *Loads the default configuration.*
- EAPI int ecore_config_save (void)
  *Saves the current configuration to the default file.*
- EAPI int ecore_config_file_load (const char *file)
  *Load the given configuration file to the local configuration.*
- EAPI int ecore_config_file_save (const char *file)
  *Saves the local configuration to the given file.*

### 6.5.1 Detailed Description

Functions that are used to load and save properties from and to files.

### 6.5.2 Function Documentation

#### 6.5.2.1 EAPI int ecore_config_file_load (const char * *file*)

Load the given configuration file to the local configuration.

**Parameters:**

***file*** Name of the file to load.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` on success. `ECORE_CONFIG_ERR_NODATA` is returned if the file cannot be loaded.

#### 6.5.2.2 EAPI int ecore_config_file_save (const char * *file*)

Saves the local configuration to the given file.

**Parameters:**

***file*** Name of the file to save to.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` is returned on success. `ECORE_CONFIG_ERR_FAIL` is returned if the data cannot be saved.

#### 6.5.2.3 EAPI int ecore_config_load (void)

Loads the default configuration.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` on success. `ECORE_CONFIG_ERR_NODATA` is returned if the file cannot be loaded.

#### 6.5.2.4 EAPI int ecore_config_save (void)

Saves the current configuration to the default file.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` is returned on success. `ECORE_CONFIG_ERR_FAIL` is returned if the data cannot be saved.

## 6.6 Process Spawning Functions

Functions that deal with spawned processes.

### Functions

- EAPI Ecore_Exe * ecore_exe_run (const char *exe_cmd, const void *data)
  *Spawns a child process.*
- EAPI Ecore_Exe * ecore_exe_pipe_run (const char *exe_cmd, Ecore_Exe_Flags flags, const void *data)
  *Spawns a child process with its stdin/out available for communication.*
- EAPI int ecore_exe_send (Ecore_Exe *exe, void *data, int size)
  *Sends data to the given child process which it recieves on stdin.*
- EAPI void ecore_exe_close_stdin (Ecore_Exe *exe)
  *The stdin of the given child process will close when the write buffer is empty.*
- EAPI void ecore_exe_auto_limits_set (Ecore_Exe *exe, int start_bytes, int end_bytes, int start_lines, int end_lines)
  *Sets the auto pipe limits for the given process handle.*
- EAPI Ecore_Exe_Event_Data * ecore_exe_event_data_get (Ecore_Exe *exe, Ecore_Exe_Flags flags)
  *Gets the auto pipe data for the given process handle.*
- EAPI void ecore_exe_tag_set (Ecore_Exe *exe, const char *tag)
  *Sets the string tag for the given process handle.*
- EAPI char * ecore_exe_tag_get (Ecore_Exe *exe)
  *Retrieves the tag attached to the given process handle.*
- EAPI void * ecore_exe_free (Ecore_Exe *exe)
  *Frees the given process handle.*
- EAPI void ecore_exe_event_data_free (Ecore_Exe_Event_Data *e)
  *Frees the given event data.*
- EAPI pid_t ecore_exe_pid_get (Ecore_Exe *exe)
  *Retrieves the process ID of the given spawned process.*
- EAPI char * ecore_exe_cmd_get (Ecore_Exe *exe)
  *Retrieves the command of the given spawned process.*
- EAPI void * ecore_exe_data_get (Ecore_Exe *exe)
  *Retrieves the data attached to the given process handle.*

### 6.6.1 Detailed Description

Functions that deal with spawned processes.

### 6.6.2 Function Documentation

#### 6.6.2.1 EAPI void ecore_exe_auto_limits_set (Ecore_Exe * *exe*, int *start_bytes*, int *end_bytes*, int *start_lines*, int *end_lines*)

Sets the auto pipe limits for the given process handle.

**Parameters:**

***exe*** The given process handle.

***start_bytes*** limit of bytes at start of output to buffer.

***end_bytes*** limit of bytes at end of output to buffer.

***start_lines*** limit of lines at start of output to buffer.

***end_lines*** limit of lines at end of output to buffer.

#### 6.6.2.2 EAPI void ecore_exe_close_stdin (Ecore_Exe * *exe*)

The stdin of the given child process will close when the write buffer is empty.

**Parameters:**

***exe*** The child process

#### 6.6.2.3 EAPI char* ecore_exe_cmd_get (Ecore_Exe * *exe*)

Retrieves the command of the given spawned process.

**Parameters:**

***exe*** Handle to the given spawned process.

**Returns:**

The command on success. NULL otherwise.

#### 6.6.2.4 EAPI void* ecore_exe_data_get (Ecore_Exe * *exe*)

Retrieves the data attached to the given process handle.

**Parameters:**

***exe*** The given process handle.

**Returns:**

The data pointer attached to `exe`.

#### 6.6.2.5 EAPI void ecore_exe_event_data_free (Ecore_Exe_Event_Data * *e*)

Frees the given event data.

**Parameters:**

***e*** The given event data.

#### 6.6.2.6 EAPI Ecore_Exe_Event_Data* ecore_exe_event_data_get (Ecore_Exe * *exe*, Ecore_Exe_Flags *flags*)

Gets the auto pipe data for the given process handle.

**Parameters:**

***exe*** The given process handle.

***flags*** Is this a ECORE_EXE_PIPE_READ or ECORE_EXE_PIPE_ERROR?

#### 6.6.2.7 EAPI void* ecore_exe_free (Ecore_Exe * *exe*)

Frees the given process handle.

Note that the process that the handle represents is unaffected by this function.

**Parameters:**

***exe*** The given process handle.

**Returns:**

The data attached to the handle when ecore_exe_run was called.

#### 6.6.2.8 EAPI pid_t ecore_exe_pid_get (Ecore_Exe * *exe*)

Retrieves the process ID of the given spawned process.

**Parameters:**

***exe*** Handle to the given spawned process.

**Returns:**

The process ID on success. `-1` otherwise.

#### 6.6.2.9 EAPI Ecore_Exe* ecore_exe_pipe_run (const char * *exe_cmd*, Ecore_Exe_Flags *flags*, const void * *data*)

Spawns a child process with its stdin/out available for communication.

This function forks and runs the given command using `/bin/sh`.

Note that the process handle is only valid until a child process terminated event is received. After all handlers for the child process terminated event have been called, the handle will be freed by Ecore.

This function does the same thing as ecore_exe_run(), but also makes the standard in and/or out as wel las stderr from the child process available for reading or writing. To write use ecore_exe_send(). To read listen to ECORE_EXE_EVENT_DATA or ECORE_EXE_EVENT_ERROR events (set up handlers). Ecore may buffer read and error data until a newline character if asked for with the `flags`. All data will be included in the events (newlines will be replaced with NULLS if line buffered). ECORE_EXE_EVENT_DATA events will only happen if the process is run with ECORE_EXE_PIPE_READ enabled in the flags. The same with the error version. Writing will only be allowed with ECORE_EXE_PIPE_WRITE enabled in the flags.

**Parameters:**

***exe_cmd*** The command to run with `/bin/sh`.

***flags*** The flag parameters for how to deal with inter-process I/O

***data*** Data to attach to the returned process handle.

**Returns:**

A process handle to the spawned process.

#### 6.6.2.10 EAPI Ecore_Exe* ecore_exe_run (const char * *exe_cmd*, const void * *data*)

Spawns a child process.

This is now just a thin wrapper around ecore_exe_pipe_run()

**Parameters:**

***exe_cmd*** The command to run with `/bin/sh`.

***data*** Data to attach to the returned process handle.

**Returns:**

A process handle to the spawned process.

#### 6.6.2.11 EAPI int ecore_exe_send (Ecore_Exe * *exe*, void * *data*, int *size*)

Sends data to the given child process which it recieves on stdin.

This function writes to a child processes standard in, with unlimited buffering. This call will never block. It may fail if the system runs out of memory.

**Parameters:**

***exe*** The child process to send to

***data*** The data to send

***size*** The size of the data to send, in bytes

**Returns:**

1 if successful, 0 on failure.

#### 6.6.2.12 EAPI char* ecore_exe_tag_get (Ecore_Exe * *exe*)

Retrieves the tag attached to the given process handle.

There is no need to free it as it just returns the internal pointer value. This value is only valid as long as the `exe` is valid or until the tag is set to something else on this `exe`.

**Parameters:**

***exe*** The given process handle.

**Returns:**

The string attached to `exe`.

#### 6.6.2.13 EAPI void ecore_exe_tag_set (Ecore_Exe * *exe*, const char * *tag*)

Sets the string tag for the given process handle.

**Parameters:**

***exe*** The given process handle.

***tag*** The string tag to set on the process handle.

## 6.7 Spawned Process Signal Functions

Functions that send signals to spawned processes.

### Functions

- EAPI void ecore_exe_pause (Ecore_Exe *exe)

  *Pauses the given process by sending it a* `SIGSTOP` *signal.*

- EAPI void ecore_exe_continue (Ecore_Exe *exe)

  *Continues the given paused process by sending it a* `SIGCONT` *signal.*

- EAPI void ecore_exe_terminate (Ecore_Exe *exe)

  *Sends the given spawned process a terminate (*`SIGTERM`*) signal.*

- EAPI void ecore_exe_kill (Ecore_Exe *exe)

  *Kills the given spawned process by sending it a* `SIGKILL` *signal.*

- EAPI void ecore_exe_signal (Ecore_Exe *exe, int num)

  *Sends a* `SIGUSR` *signal to the given spawned process.*

- EAPI void ecore_exe_hup (Ecore_Exe *exe)

  *Sends a* `SIGHUP` *signal to the given spawned process.*

### 6.7.1 Detailed Description

Functions that send signals to spawned processes.

### 6.7.2 Function Documentation

#### 6.7.2.1 EAPI void ecore_exe_continue (Ecore_Exe * *exe*)

Continues the given paused process by sending it a `SIGCONT` signal.

**Parameters:**

***exe*** Process handle to the given process.

#### 6.7.2.2 EAPI void ecore_exe_hup (Ecore_Exe * *exe*)

Sends a SIGHUP signal to the given spawned process.

**Parameters:**

***exe*** Process handle to the given process.

#### 6.7.2.3 EAPI void ecore_exe_kill (Ecore_Exe * *exe*)

Kills the given spawned process by sending it a SIGKILL signal.

**Parameters:**

***exe*** Process handle to the given process.

#### 6.7.2.4 EAPI void ecore_exe_pause (Ecore_Exe * *exe*)

Pauses the given process by sending it a SIGSTOP signal.

**Parameters:**

***exe*** Process handle to the given process.

#### 6.7.2.5 EAPI void ecore_exe_signal (Ecore_Exe * *exe*, int *num*)

Sends a SIGUSR signal to the given spawned process.

**Parameters:**

***exe*** Process handle to the given process.

***num*** The number user signal to send. Must be either 1 or 2, or the signal will be ignored.

#### 6.7.2.6 EAPI void ecore_exe_terminate (Ecore_Exe * *exe*)

Sends the given spawned process a terminate (SIGTERM) signal.

**Parameters:**

***exe*** Process handle to the given process.

## 6.8 Hash Creation Functions

Functions that create hash tables.

### Functions

- EAPI Ecore_Hash * ecore_hash_new (Ecore_Hash_Cb hash_func, Ecore_Compare_Cb compare)

  *Creates and initializes a new hash.*

- EAPI int ecore_hash_init (Ecore_Hash *hash, Ecore_Hash_Cb hash_func, Ecore_Compare_Cb compare)

  *Initializes the given hash.*

### 6.8.1 Detailed Description

Functions that create hash tables.

### 6.8.2 Function Documentation

#### 6.8.2.1 EAPI int ecore_hash_init (Ecore_Hash * *hash*, Ecore_Hash_Cb *hash_func*, Ecore_Compare_Cb *compare*)

Initializes the given hash.

**Parameters:**

***hash*** The given hash.

***hash_func*** The function used for hashing node keys.

***compare*** The function used for comparing node keys.

**Returns:**

`TRUE` on success, `FALSE` on an error.

#### 6.8.2.2 EAPI Ecore_Hash* ecore_hash_new (Ecore_Hash_Cb *hash_func*, Ecore_Compare_Cb *compare*)

Creates and initializes a new hash.

**Parameters:**

***hash_func*** The function for determining hash position.

***compare*** The function for comparing node keys.

**Returns:**

`NULL` on error, a new hash on success.

## 6.9 Hash Destruction Functions

Functions that destroy hash tables and their contents.

### Functions

- EAPI int ecore_hash_set_free_key (Ecore_Hash *hash, Ecore_Free_Cb function)
  *Sets the function to destroy the keys of the given hash.*
- EAPI int ecore_hash_set_free_value (Ecore_Hash *hash, Ecore_Free_Cb function)
  *Sets the function to destroy the values in the given hash.*
- EAPI void ecore_hash_destroy (Ecore_Hash *hash)
  *Frees the hash table and the data contained inside it.*
- EAPI int ecore_hash_count (Ecore_Hash *hash)
  *Counts the number of nodes in a hash table.*

### 6.9.1 Detailed Description

Functions that destroy hash tables and their contents.

### 6.9.2 Function Documentation

#### 6.9.2.1 EAPI int ecore_hash_count (Ecore_Hash * *hash*)

Counts the number of nodes in a hash table.

**Parameters:**

***hash*** The hash table to count current nodes.

**Returns:**

The number of nodes in the hash.

#### 6.9.2.2 EAPI void ecore_hash_destroy (Ecore_Hash * *hash*)

Frees the hash table and the data contained inside it.

**Parameters:**

***hash*** The hash table to destroy.

**Returns:**

TRUE on success, FALSE on error.

#### 6.9.2.3 EAPI int ecore_hash_set_free_key (Ecore_Hash * *hash*, Ecore_Free_Cb *function*)

Sets the function to destroy the keys of the given hash.

**Parameters:**

***hash*** The given hash.

***function*** The function used to free the node keys.

**Returns:**

TRUE on success, FALSE on error.

#### 6.9.2.4 EAPI int ecore_hash_set_free_value (Ecore_Hash * *hash*, Ecore_Free_Cb *function*)

Sets the function to destroy the values in the given hash.

**Parameters:**

***hash*** The given hash.

***function*** The function that will free the node values.

**Returns:**

TRUE on success, FALSE on error

## 6.10 Hash Data Functions

Functions that set, access and delete values from the hash tables.

### Functions

- EAPI int ecore_hash_set (Ecore_Hash *hash, void *key, void *value)
  *Sets a key-value pair in the given hash table.*
- EAPI int ecore_hash_set_hash (Ecore_Hash *hash, Ecore_Hash *set)
  *Sets all key-value pairs from set in the given hash table.*
- EAPI void * ecore_hash_get (Ecore_Hash *hash, const void *key)
  *Retrieves the value associated with the given key from the given hash table.*
- EAPI void * ecore_hash_remove (Ecore_Hash *hash, const void *key)
  *Removes the value associated with the given key in the given hash table.*
- EAPI void * ecore_hash_find (Ecore_Hash *hash, Ecore_Compare_Cb compare, const void *value)
  *Retrieves the first value that matches table.*

### 6.10.1 Detailed Description

Functions that set, access and delete values from the hash tables.

### 6.10.2 Function Documentation

#### 6.10.2.1 EAPI void* ecore_hash_find (Ecore_Hash * *hash*, Ecore_Compare_Cb *compare*, const void * *value*)

Retrieves the first value that matches table.

**Parameters:**

***hash*** The given hash table.

***key*** The key to search for.

**Returns:**

The value corresponding to key on success, `NULL` otherwise.

#### 6.10.2.2 EAPI void* ecore_hash_get (Ecore_Hash * *hash*, const void * *key*)

Retrieves the value associated with the given key from the given hash table.

**Parameters:**

***hash*** The given hash table.

***key*** The key to search for.

**Returns:**

The value corresponding to key on success, `NULL` otherwise.

#### 6.10.2.3 EAPI void* ecore_hash_remove (Ecore_Hash * *hash*, const void * *key*)

Removes the value associated with the given key in the given hash table.

**Parameters:**

***hash*** The given hash table.

***key*** The key to search for.

**Returns:**

The value corresponding to the key on success. `NULL` is returned if there is an error.

#### 6.10.2.4 EAPI int ecore_hash_set (Ecore_Hash * *hash*, void * *key*, void * *value*)

Sets a key-value pair in the given hash table.

**Parameters:**

***hash*** The given hash table.

***key*** The key.

***value*** The value.

**Returns:**

`TRUE` if successful, `FALSE` if not.

#### 6.10.2.5 EAPI int ecore_hash_set_hash (Ecore_Hash * *hash*, Ecore_Hash * *set*)

Sets all key-value pairs from set in the given hash table.

**Parameters:**

***hash*** The given hash table.

***set*** The hash table to import.

**Returns:**

`TRUE` if successful, `FALSE` if not.

## 6.11 Hash Traverse Functions

Functions that iterate through hash tables.

### Functions

- EAPI int ecore_hash_for_each_node (Ecore_Hash *hash, Ecore_For_Each for_each_-func, void *user_data)

  *Runs the* `for_each_func` *function on each entry in the given hash.*

- EAPI Ecore_List * ecore_hash_keys (Ecore_Hash *hash)

  *Retrieves an ecore_list of all keys in the given hash.*

### 6.11.1 Detailed Description

Functions that iterate through hash tables.

### 6.11.2 Function Documentation

#### 6.11.2.1 EAPI int ecore_hash_for_each_node (Ecore_Hash * *hash*, Ecore_For_Each *for_each_func*, void * *user_data*)

Runs the `for_each_func` function on each entry in the given hash.

**Parameters:**

***hash*** The given hash.

***for_each_func*** The function that each entry is passed to.

***user_data*** a pointer passed to calls of for_each_func

**Returns:**

TRUE on success, FALSE otherwise.

#### 6.11.2.2 EAPI Ecore_List* ecore_hash_keys (Ecore_Hash * *hash*)

Retrieves an ecore_list of all keys in the given hash.

**Parameters:**

***hash*** The given hash.

**Returns:**

new ecore_list on success, NULL otherwise

## 6.12 List Creation/Destruction Functions

Functions that create, initialize and destroy Ecore_Lists.

### Functions

- EAPI Ecore_List * ecore_list_new ()
  *Create and initialize a new list.*
- EAPI int ecore_list_init (Ecore_List *list)
  *Initialize a list to some sane starting values.*
- EAPI void ecore_list_destroy (Ecore_List *list)
  *Free a list and all of it's nodes.*

### 6.12.1 Detailed Description

Functions that create, initialize and destroy Ecore_Lists.

### 6.12.2 Function Documentation

#### 6.12.2.1 EAPI void ecore_list_destroy (Ecore_List * *list*)

Free a list and all of it's nodes.

**Parameters:**

***list*** The list to be freed.

#### 6.12.2.2 EAPI int ecore_list_init (Ecore_List * *list*)

Initialize a list to some sane starting values.

**Parameters:**

***list*** The list to initialize.

**Returns:**

`TRUE` if successful, `FALSE` if an error occurs.

#### 6.12.2.3 EAPI Ecore_List* ecore_list_new (void)

Create and initialize a new list.

**Returns:**

A new initialized list on success, `NULL` on failure.

## 6.13 List Item Adding Functions

Functions that are used to add nodes to an Ecore_List.

### Functions

- EAPI int ecore_list_append (Ecore_List *list, void *data)
  *Append data to the list.*
- EAPI int ecore_list_prepend (Ecore_List *list, void *data)
  *Prepend data to the beginning of the list.*
- EAPI int ecore_list_insert (Ecore_List *list, void *data)
  *Insert data in front of the current point in the list.*
- EAPI int ecore_list_append_list (Ecore_List *list, Ecore_List *append)
  *Append a list to the list.*
- EAPI int ecore_list_prepend_list (Ecore_List *list, Ecore_List *prepend)
  *Prepend a list to the beginning of the list.*

### 6.13.1 Detailed Description

Functions that are used to add nodes to an Ecore_List.

### 6.13.2 Function Documentation

#### 6.13.2.1 EAPI int ecore_list_append (Ecore_List * *list*, void * *data*) [inline]

Append data to the list.

**Parameters:**

***list*** The list.

***data*** The data to append.

**Returns:**

`FALSE` if an error occurs, `TRUE` if appended successfully

#### 6.13.2.2 EAPI int ecore_list_append_list (Ecore_List * *list*, Ecore_List * *append*)

Append a list to the list.

**Parameters:**

***list*** The list.

***append*** The list to append.

**Returns:**

`FALSE` if an error occurs, `TRUE` if appended successfully

#### 6.13.2.3 EAPI int ecore_list_insert (Ecore_List * *list*, void * *data*) [inline]

Insert data in front of the current point in the list.

**Parameters:**

***list*** The list to hold the inserted `data`.

***data*** The data to insert into `list`.

**Returns:**

`FALSE` if there is an error, `TRUE` on success

#### 6.13.2.4 EAPI int ecore_list_prepend (Ecore_List * *list*, void * *data*) [inline]

Prepend data to the beginning of the list.

**Parameters:**

***list*** The list.

***data*** The data to prepend.

**Returns:**

`FALSE` if an error occurs, `TRUE` if prepended successfully.

#### 6.13.2.5 EAPI int ecore_list_prepend_list (Ecore_List * *list*, Ecore_List * *prepend*)

Prepend a list to the beginning of the list.

**Parameters:**

***list*** The list.

***prepend*** The list to prepend.

**Returns:**

`FALSE` if an error occurs, `TRUE` if prepended successfully.

## 6.14 List Item Removing Functions

Functions that remove nodes from an Ecore_List.

### Functions

- EAPI void * ecore_list_remove (Ecore_List *list)
  *Remove the current item from the list.*
- EAPI int ecore_list_remove_destroy (Ecore_List *list)
  *Remove and free the data in lists current position.*
- EAPI void * ecore_list_remove_first (Ecore_List *list)
  *Remove the first item from the list.*
- EAPI void * ecore_list_remove_last (Ecore_List *list)
  *Remove the last item from the list.*

### 6.14.1 Detailed Description

Functions that remove nodes from an Ecore_List.

### 6.14.2 Function Documentation

#### 6.14.2.1 EAPI void* ecore_list_remove (Ecore_List * *list*) [inline]

Remove the current item from the list.

**Parameters:**

***list*** The list to remove the current item

**Returns:**

A pointer to the removed data on success, NULL on failure.

#### 6.14.2.2 EAPI int ecore_list_remove_destroy (Ecore_List * *list*)

Remove and free the data in lists current position.

**Parameters:**

***list*** The list to remove and free the current item.

**Returns:**

`TRUE` on success, `FALSE` on error

#### 6.14.2.3 EAPI void* ecore_list_remove_first (Ecore_List * *list*) [inline]

Remove the first item from the list.

**Parameters:**

***list*** The list to remove the current item

**Returns:**

Returns a pointer to the removed data on success, `NULL` on failure.

#### 6.14.2.4 EAPI void* ecore_list_remove_last (Ecore_List * *list*) [inline]

Remove the last item from the list.

**Parameters:**

***list*** The list to remove the last node from

**Returns:**

A pointer to the removed data on success, `NULL` on failure.

## 6.15 List Traversal Functions

Functions that can be used to traverse an Ecore_List.

### Functions

- EAPI void * ecore_list_goto_index (Ecore_List *list, int index)

  *Make the current item the item with the given index number.*

- EAPI void * ecore_list_goto (Ecore_List *list, void *data)

  *Make the current item the node that contains* `data`.

- EAPI void * ecore_list_goto_first (Ecore_List *list)

  *Make the current item the first item in the list.*

- EAPI void * ecore_list_goto_last (Ecore_List *list)

  *Make the current item the last item in the list.*

- EAPI int ecore_list_for_each (Ecore_List *list, Ecore_For_Each function, void *user_data)

  *Execute function for each node in* `list`.

### 6.15.1 Detailed Description

Functions that can be used to traverse an Ecore_List.

### 6.15.2 Function Documentation

#### 6.15.2.1 EAPI int ecore_list_for_each (Ecore_List * *list*, Ecore_For_Each *function*, void * *user_data*)

Execute function for each node in `list`.

**Parameters:**

***list*** The list.

***function*** The function to pass each node from `list` to.

**Returns:**

Returns `TRUE` on success, `FALSE` on failure.

#### 6.15.2.2 EAPI void* ecore_list_goto (Ecore_List * *list*, void * *data*) [inline]

Make the current item the node that contains `data`.

**Parameters:**

***list*** The list.

***data*** The data to find.

**Returns:**

A pointer to `data` on success, `NULL` on failure.

#### 6.15.2.3 EAPI void* ecore_list_goto_first (Ecore_List * *list*) [inline]

Make the current item the first item in the list.

**Parameters:**

***list*** The list.

**Returns:**

A pointer to the first item on success, `NULL` on failure

#### 6.15.2.4 EAPI void* ecore_list_goto_index (Ecore_List * *list*, int *index*) [inline]

Make the current item the item with the given index number.

**Parameters:**

***list*** The list.

***index*** The position to move the current item.

**Returns:**

A pointer to new current item on success, `NULL` on failure.

#### 6.15.2.5 EAPI void* ecore_list_goto_last (Ecore_List * *list*) [inline]

Make the current item the last item in the list.

**Parameters:**

***list*** The list.

**Returns:**

A pointer to the last item on success, `NULL` on failure.

## 6.16 List Node Functions

Functions that are used in the creation, maintenance and destruction of Ecore_List nodes.

### Functions

- EAPI Ecore_List_Node * ecore_list_node_new ()

  *Allocates and initializes a new list node.*

- EAPI int ecore_list_node_destroy (Ecore_List_Node *node, Ecore_Free_Cb free_func)

  *Calls the function to free the data and the node.*

### 6.16.1 Detailed Description

Functions that are used in the creation, maintenance and destruction of Ecore_List nodes.

### 6.16.2 Function Documentation

#### 6.16.2.1 EAPI int ecore_list_node_destroy (Ecore_List_Node * *node*, Ecore_Free_Cb *free_func*)

Calls the function to free the data and the node.

**Parameters:**

***node*** Node to destroy.

***free_func*** Function to call if `node` points to data to free.

**Returns:**

`TRUE`.

#### 6.16.2.2 EAPI Ecore_List_Node* ecore_list_node_new (void)

Allocates and initializes a new list node.

**Returns:**

A new Ecore_List_Node on success, `NULL` otherwise.

## 6.17 Doubly Linked List Creation/Destruction Functions

Functions used to create, initialize and destroy `Ecore_DLists`.

### Functions

- EAPI Ecore_DList * ecore_dlist_new ()
  *Creates and initialises a new doubly linked list.*
- EAPI int ecore_dlist_init (Ecore_DList *list)
  *Initialises a list to some sane starting values.*
- EAPI void ecore_dlist_destroy (Ecore_DList *list)
  *Frees a doubly linked list and all of its nodes.*
- EAPI int ecore_dlist_set_free_cb (Ecore_DList *list, Ecore_Free_Cb free_func)
  *Sets the function used for freeing data stored in a doubly linked list.*

### 6.17.1 Detailed Description

Functions used to create, initialize and destroy `Ecore_DLists`.

### 6.17.2 Function Documentation

#### 6.17.2.1 EAPI void ecore_dlist_destroy (Ecore_DList * *list*)

Frees a doubly linked list and all of its nodes.

**Parameters:**

***list*** The doubly linked list to be freed.

#### 6.17.2.2 EAPI int ecore_dlist_init (Ecore_DList * *list*)

Initialises a list to some sane starting values.

**Parameters:**

***list*** The doubly linked list to initialise.

**Returns:**

`TRUE` if successful, `FALSE` if an error occurs.

#### 6.17.2.3 EAPI Ecore_DList* ecore_dlist_new (void)

Creates and initialises a new doubly linked list.

**Returns:**

A new initialised doubly linked list on success, `NULL` on failure.

#### 6.17.2.4 EAPI int ecore_dlist_set_free_cb (Ecore_DList * *list*, Ecore_Free_Cb *free_func*)

Sets the function used for freeing data stored in a doubly linked list.

**Parameters:**

***list*** The doubly linked list that will use this function when nodes are destroyed.

***free_func*** The function that will free the key data

**Returns:**

`TRUE` on success, `FALSE` on failure.

## 6.18 Doubly Linked List Adding Functions

Functions that are used to add nodes to an Ecore_DList.

### Functions

- EAPI int ecore_dlist_append (Ecore_DList *list, void *data)
  *Appends data to the given doubly linked list.*
- EAPI int ecore_dlist_prepend (Ecore_DList *list, void *data)
  *Adds data to the very beginning of the given doubly linked list.*
- EAPI int ecore_dlist_insert (Ecore_DList *list, void *data)
  *Inserts data at the current point in the given doubly linked list.*
- EAPI int ecore_dlist_append_list (Ecore_DList *list, Ecore_DList *append)
  *Appends a list to the given doubly linked list.*
- EAPI int ecore_dlist_prepend_list (Ecore_DList *list, Ecore_DList *prepend)
  *Adds a list to the very beginning of the given doubly linked list.*

### 6.18.1 Detailed Description

Functions that are used to add nodes to an Ecore_DList.

### 6.18.2 Function Documentation

#### 6.18.2.1 EAPI int ecore_dlist_append (Ecore_DList * *list*, void * *data*)

Appends data to the given doubly linked list.

**Parameters:**

***list*** The given doubly linked list.

***data*** The data to append.

**Returns:**

`TRUE` if the data is successfully appended, `FALSE` otherwise.

#### 6.18.2.2 EAPI int ecore_dlist_append_list (Ecore_DList * *list*, Ecore_DList * *append*)

Appends a list to the given doubly linked list.

**Parameters:**

***list*** The given doubly linked list.

***append*** The list to append.

**Returns:**

`TRUE` if the data is successfully appended, `FALSE` otherwise.

#### 6.18.2.3 EAPI int ecore_dlist_insert (Ecore_DList * *list*, void * *data*)

Inserts data at the current point in the given doubly linked list.

**Parameters:**

***list*** The given doubly linked list.

***data*** The data to be inserted.

**Returns:**

`TRUE` on success, `FALSE` otherwise.

#### 6.18.2.4 EAPI int ecore_dlist_prepend (Ecore_DList * *list*, void * *data*)

Adds data to the very beginning of the given doubly linked list.

**Parameters:**

***list*** The given doubly linked list.

***data*** The data to prepend.

**Returns:**

`TRUE` if the data is successfully prepended, `FALSE` otherwise.

#### 6.18.2.5 EAPI int ecore_dlist_prepend_list (Ecore_DList * *list*, Ecore_DList * *prepend*)

Adds a list to the very beginning of the given doubly linked list.

**Parameters:**

***list*** The given doubly linked list.

***prepend*** The list to prepend.

**Returns:**

`TRUE` if the data is successfully prepended, `FALSE` otherwise.

## 6.19 Doubly Linked List Removing Functions

Functions that remove nodes from an `Ecore_DList`.

### Functions

- EAPI void * ecore_dlist_remove (Ecore_DList *list)
  *Removes the current item from the given doubly linked list.*
- EAPI void * ecore_dlist_remove_first (Ecore_DList *list)
  *Removes the first item from the given doubly linked list.*
- EAPI int ecore_dlist_remove_destroy (Ecore_DList *list)
  *Removes and frees the data at the current position in the given doubly linked list.*
- EAPI void * ecore_dlist_remove_last (Ecore_DList *list)
  *Removes the last item from the given doubly linked list.*

### 6.19.1 Detailed Description

Functions that remove nodes from an `Ecore_DList`.

### 6.19.2 Function Documentation

#### 6.19.2.1 EAPI void* ecore_dlist_remove (Ecore_DList * *list*)

Removes the current item from the given doubly linked list.

**Parameters:**

***list*** The given doubly linked list.

**Returns:**

A pointer to the removed data on success, `NULL` otherwise.

#### 6.19.2.2 EAPI int ecore_dlist_remove_destroy (Ecore_DList * *list*)

Removes and frees the data at the current position in the given doubly linked list.

**Parameters:**

***list*** The given doubly linked list.

**Returns:**

TRUE on success, FALSE otherwise.

#### 6.19.2.3 EAPI void* ecore_dlist_remove_first (Ecore_DList * *list*)

Removes the first item from the given doubly linked list.

**Parameters:**

***list*** The given doubly linked list.

**Returns:**

A pointer to the removed data on success, NULL on failure.

#### 6.19.2.4 EAPI void* ecore_dlist_remove_last (Ecore_DList * *list*)

Removes the last item from the given doubly linked list.

**Parameters:**

***list*** The given doubly linked list.

**Returns:**

A pointer to the removed data on success, NULL otherwise.

## 6.20 Main Loop Functions

These functions control the Ecore event handling loop.

### Functions

- EAPI void ecore_main_loop_iterate (void)

  *Runs a single iteration of the main loop to process everything on the queue.*

- EAPI void ecore_main_loop_begin (void)

  *Runs the application main loop.*

- EAPI void ecore_main_loop_quit (void)

  *Quits the main loop once all the events currently on the queue have been processed.*

### 6.20.1 Detailed Description

These functions control the Ecore event handling loop.

This loop is designed to work on embedded systems all the way to large and powerful mutli-cpu workstations.

It serialises all system signals and events into a single event queue, that can be easily processed without needing to worry about concurrency. A properly written, event-driven program using this kind of programming does not need threads. It makes the program very robust and easy to follow.

Here is an example of simple program and its basic event loop flow:

For examples of setting up and using a main loop, see event_handler_example::c and timer_example::c.

### 6.20.2 Function Documentation

#### 6.20.2.1 EAPI void ecore_main_loop_begin (void)

Runs the application main loop.

This function will not return until ecore_main_loop_quit is called.

## 6.21 File Event Handling Functions

Functions that deal with file descriptor handlers.

### Functions

- EAPI Ecore_Fd_Handler * ecore_main_fd_handler_add (int fd, Ecore_Fd_Handler_Flags flags, int(*func)(void *data, Ecore_Fd_Handler *fd_handler), const void *data, int(*buf_func)(void *buf_data, Ecore_Fd_Handler *fd_handler), const void *buf_data)

  *Adds a callback for activity on the given file descriptor.*

- EAPI void * ecore_main_fd_handler_del (Ecore_Fd_Handler *fd_handler)

  *Deletes the given FD handler.*

- EAPI int ecore_main_fd_handler_fd_get (Ecore_Fd_Handler *fd_handler)

  *Retrieves the file descriptor that the given handler is handling.*

- EAPI int ecore_main_fd_handler_active_get (Ecore_Fd_Handler *fd_handler, Ecore_Fd_Handler_Flags flags)

  *Return if read, write or error, or a combination thereof, is active on the file descriptor of the given FD handler.*

- EAPI void ecore_main_fd_handler_active_set (Ecore_Fd_Handler *fd_handler, Ecore_Fd_Handler_Flags flags)

  *Set what active streams the given FD handler should be monitoring.*

### 6.21.1 Detailed Description

Functions that deal with file descriptor handlers.

### 6.21.2 Function Documentation

#### 6.21.2.1 EAPI int ecore_main_fd_handler_active_get (Ecore_Fd_Handler * *fd_handler*, Ecore_Fd_Handler_Flags *flags*)

Return if read, write or error, or a combination thereof, is active on the file descriptor of the given FD handler.

**Parameters:**

***fd_handler*** The given FD handler.

***flags*** The flags, `ECORE_FD_READ`, `ECORE_FD_WRITE` or `ECORE_FD_ERROR` to query.

**Returns:**

1 if any of the given flags are active. 0 otherwise.

#### 6.21.2.2 EAPI void ecore_main_fd_handler_active_set (Ecore_Fd_Handler * *fd_handler*, Ecore_Fd_Handler_Flags *flags*)

Set what active streams the given FD handler should be monitoring.

**Parameters:**

***fd_handler*** The given FD handler.

***flags*** The flags to be watching.

#### 6.21.2.3 EAPI Ecore_Fd_Handler* ecore_main_fd_handler_add (int *fd*, Ecore_Fd_Handler_Flags *flags*, int(*)(void *data, Ecore_Fd_Handler *fd_handler) *func*, const void * *data*, int(*)(void *buf_data, Ecore_Fd_Handler *fd_handler) *buf_func*, const void * *buf_data*)

Adds a callback for activity on the given file descriptor.

`func` will be called during the execution of ecore_main_loop_begin when the file descriptor is available for reading, or writing, or both.

Normally the return value from the `func` is "zero means this handler is finished and can be deleted" as is usual for handler callbacks. However, if the `buf_func` is supplied, then the return value from the `func` is "non zero means the handler should be called again in a tight loop".

`buf_func` is called during event loop handling to check if data that has been read from the file descriptor is in a buffer and is available to read. Some systems (notably xlib) handle their own buffering, and would otherwise not work with select(). These systems should use a `buf_func`. This is a most annoying hack, only ecore_x uses it, so refer to that for an example. NOTE - `func` should probably return "one" always if `buf_func` is used, to avoid confusion with the other return value semantics.

**Parameters:**

***fd*** The file descriptor to watch.

***flags*** To watch it for read (`ECORE_FD_READ`) and/or (`ECORE_FD_WRITE`) write ability. `ECORE_FD_ERROR`

***func*** The callback function.

***data*** The data to pass to the callback.

***buf_func*** The function to call to check if any data has been buffered and already read from the fd. Can be `NULL`.

***buf_data*** The data to pass to the `buf_func` function.

**Returns:**

A fd handler handle if successful. `NULL` otherwise.

#### 6.21.2.4 EAPI void* ecore_main_fd_handler_del (Ecore_Fd_Handler * *fd_handler*)

Deletes the given FD handler.

**Parameters:**

***fd_handler*** The given FD handler.

**Returns:**

The data pointer set using ecore_main_fd_handler_add, for `fd_handler` on success. `NULL` otherwise.

#### 6.21.2.5 EAPI int ecore_main_fd_handler_fd_get (Ecore_Fd_Handler * *fd_handler*)

Retrieves the file descriptor that the given handler is handling.

**Parameters:**

***fd_handler*** The given FD handler.

**Returns:**

The file descriptor the handler is watching.

## 6.22 Path Group Functions

Functions that make it easier to find a file in a set of search paths.

### Functions

- EAPI int ecore_path_group_new (char *group_name)
  *Creates a new path group.*
- EAPI void ecore_path_group_del (int group_id)
  *Destroys a previously created path group.*
- EAPI void ecore_path_group_add (int group_id, char *path)
  *Adds a directory to be searched for files.*
- EAPI void ecore_path_group_remove (int group_id, char *path)
  *Removes the given directory from the given group.*
- EAPI char * ecore_path_group_find (int group_id, char *name)
  *Finds a file in a group of paths.*
- EAPI Ecore_List * ecore_path_group_available (int group_id)
  *Retrieves a list of all available files in the given path.*

### 6.22.1 Detailed Description

Functions that make it easier to find a file in a set of search paths.

**Todo**

Give this a better description.

### 6.22.2 Function Documentation

#### 6.22.2.1 EAPI void ecore_path_group_add (int *group_id*, char * *path*)

Adds a directory to be searched for files.

**Parameters:**

***group_id*** The unique identifier for the group to add the path.
***path*** The new path to be added to the group.

#### 6.22.2.2 EAPI Ecore_List* ecore_path_group_available (int *group_id*)

Retrieves a list of all available files in the given path.

**Parameters:**

***group_id*** The identifier for the given path.

**Returns:**

A pointer to a newly allocated list of all files found in the paths identified by `group_id`. `NULL` otherwise.

#### 6.22.2.3 EAPI void ecore_path_group_del (int *group_id*)

Destroys a previously created path group.

**Parameters:**

***group_id*** The unique identifier for the group.

#### 6.22.2.4 EAPI char* ecore_path_group_find (int *group_id*, char * *name*)

Finds a file in a group of paths.

**Parameters:**

***group_id*** The path group id to search.

***name*** The name of the file to find.

**Returns:**

A pointer to a newly allocated path location of the found file on success. `NULL` on failure.

#### 6.22.2.5 EAPI int ecore_path_group_new (char * *group_name*)

Creates a new path group.

**Parameters:**

***group_name*** The name of the new group.

**Returns:**

`0` on error, the integer id of the new group on success.

#### 6.22.2.6 EAPI void ecore_path_group_remove (int *group_id*, char * *path*)

Removes the given directory from the given group.

**Parameters:**

***group_id*** The identifier for the given group.

***path*** The path of the directory to be removed.

## 6.23 Plugin Functions

Functions that load modules of compiled code into memory.

### Functions

- EAPI Ecore_Plugin * ecore_plugin_load (int group_id, const char *plugin_name)
  *Loads the specified plugin from the specified path group.*
- EAPI void ecore_plugin_unload (Ecore_Plugin *plugin)
  *Unloads the given plugin from memory.*

### 6.23.1 Detailed Description

Functions that load modules of compiled code into memory.

### 6.23.2 Function Documentation

#### 6.23.2.1 EAPI Ecore_Plugin* ecore_plugin_load (int *group_id*, const char * *plugin_name*)

Loads the specified plugin from the specified path group.

**Parameters:**

***group_id*** The path group to search for the plugin to load

***plugin_name*** The name of the plugin to load.

**Returns:**

A pointer to the newly loaded plugin on success, `NULL` on failure.

#### 6.23.2.2 EAPI void ecore_plugin_unload (Ecore_Plugin * *plugin*)

Unloads the given plugin from memory.

**Parameters:**

***plugin*** The given plugin.

## 6.24 String Instance Functions

These functions allow you to store one copy of a string, and use it throughout your program.

### Functions

- EAPI const char * ecore_string_instance (const char *string)
  *Retrieves an instance of a string for use in an ecore program.*
- EAPI void ecore_string_release (const char *string)
  *Notes that the given string has lost an instance.*

### 6.24.1 Detailed Description

These functions allow you to store one copy of a string, and use it throughout your program.

### 6.24.2 Function Documentation

#### 6.24.2.1 EAPI const char* ecore_string_instance (const char * *string*)

Retrieves an instance of a string for use in an ecore program.

**Parameters:**

***string*** The string to retrieve an instance of.

**Returns:**

A pointer to an instance of the string on success. `NULL` on failure.

#### 6.24.2.2 EAPI void ecore_string_release (const char * *string*)

Notes that the given string has lost an instance.

It will free the string if no other instances are left.

**Parameters:**

***string*** The given string.

## 6.25 Ecore Time Functions

Functions that deal with time.

### Functions

- EAPI double ecore_time_get (void)

  *Retrieves the current system time as a floating point value in seconds.*

- EAPI Ecore_Timer * ecore_timer_add (double in, int(*func)(void *data), const void *data)

  *Creates a timer to call the given function in the given period of time.*

- EAPI void * ecore_timer_del (Ecore_Timer *timer)

  *Delete the specified timer from the timer list.*

- EAPI void ecore_timer_interval_set (Ecore_Timer *timer, double in)

  *Change the interval the timer ticks of.*

### 6.25.1 Detailed Description

Functions that deal with time.

These functions include those that simply retrieve it in a given format, and those that create events based on it.

### 6.25.2 Function Documentation

#### 6.25.2.1 EAPI double ecore_time_get (void)

Retrieves the current system time as a floating point value in seconds.

**Returns:**

The number of seconds since 12.00AM 1st January 1970.

#### 6.25.2.2 EAPI Ecore_Timer* ecore_timer_add (double *in*, int(*)(void *data) *func*, const void * *data*)

Creates a timer to call the given function in the given period of time.

**Parameters:**

***in*** The interval in seconds.

***func*** The given function. If `func` returns 1, the timer is rescheduled for the next interval `in`.

***data*** Data to pass to `func` when it is called.

**Returns:**

A timer object on success. `NULL` on failure.

This function adds a timer and returns its handle on success and NULL on failure. The function `func` will be called every @ seconds. The function will be passed the `data` pointer as its parameter.

When the timer `func` is called, it must return a value of either 1 or 0. If it returns 1, it will be called again at the next tick, or if it returns 0 it will be deleted automatically making any references/handles for it invalid.

#### 6.25.2.3 EAPI void* ecore_timer_del (Ecore_Timer * *timer*)

Delete the specified timer from the timer list.

**Parameters:**

***timer*** The timer to delete.

**Returns:**

The data pointer set for the timer when ecore_timer_add was called. `NULL` is returned if the function is unsuccessful.

Note: `timer` must be a valid handle. If the timer function has already returned 0, the handle is no longer valid (and does not need to be delete).

#### 6.25.2.4 EAPI void ecore_timer_interval_set (Ecore_Timer * *timer*, double *in*)

Change the interval the timer ticks of.

If set during a timer call, this will affect the next interval.

**Parameters:**

***timer*** The timer to change.

***in*** The interval in seconds.

## 6.26 Ecore Connection Library Functions

Utility functions that set up and shut down the Ecore Connection library.

### Functions

- EAPI int ecore_con_init (void)
  *Initialises the Ecore_ Con library.*
- EAPI int ecore_con_shutdown (void)
  *Shuts down the Ecore_ Con library.*

### 6.26.1 Detailed Description

Utility functions that set up and shut down the Ecore Connection library.

### 6.26.2 Function Documentation

#### 6.26.2.1 EAPI int ecore_con_init (void)

Initialises the Ecore_Con library.

**Returns:**

Number of times the library has been initialised without being shut down.

#### 6.26.2.2 EAPI int ecore_con_shutdown (void)

Shuts down the Ecore_Con library.

**Returns:**

Number of times the library has been initialised without being shut down.

## 6.27 Ecore Connection Server Functions

Functions that operate on Ecore server objects.

### Functions

- EAPI Ecore_Con_Server * ecore_con_server_add (Ecore_Con_Type compl_type, const char *name, int port, const void *data)

  *Creates a server to listen for connections.*

- EAPI Ecore_Con_Server * ecore_con_server_connect (Ecore_Con_Type compl_type, const char *name, int port, const void *data)

  *Creates a server object to represent the server listening at the given port.*

- EAPI void * ecore_con_server_del (Ecore_Con_Server *svr)

  *Closes the connection and frees the given server.*

- EAPI void * ecore_con_server_data_get (Ecore_Con_Server *svr)

  *Retrieves the data associated with the given server.*

- EAPI int ecore_con_server_connected_get (Ecore_Con_Server *svr)

  *Retrieves whether the given server is currently connected.*

- EAPI Ecore_List * ecore_con_server_clients_get (Ecore_Con_Server *svr)

  *Retrieves the current list of clients.*

- EAPI int ecore_con_server_send (Ecore_Con_Server *svr, const void *data, int size)

  *Sends the given data to the given server.*

- EAPI void ecore_con_server_client_limit_set (Ecore_Con_Server *svr, int client_limit, char reject_excess_clients)

  *Sets a limit on the number of clients that can be handled concurrently by the given server, and a policy on what to do if excess clients try to connect.*

- EAPI char * ecore_con_server_ip_get (Ecore_Con_Server *svr)

  *Gets the IP address of a server that has been connected to.*

- EAPI void ecore_con_server_flush (Ecore_Con_Server *svr)

  *Flushes all pending data to the given server.*

### 6.27.1 Detailed Description

Functions that operate on Ecore server objects.

## 6.27.2 Function Documentation

### 6.27.2.1 EAPI Ecore_Con_Server* ecore_con_server_add (Ecore_Con_Type *compl_type*, const char * *name*, int *port*, const void * *data*)

Creates a server to listen for connections.

The socket on which the server listens depends on the connection type:

- If *compl_type* is `ECORE_CON_LOCAL_USER`, the server will listen on the Unix socket "~/.ecore/[name]/[port]".
- If *compl_type* is `ECORE_CON_LOCAL_SYSTEM`, the server will listen on Unix socket "/tmp/.ecore_service|[name]|[port]".
- If *compl_type* is `ECORE_CON_REMOTE_SYSTEM`, the server will listen on TCP port `port`.

**Parameters:**

***compl_type*** The connection type.

***name*** Name to associate with the socket. It is used when generating the socket name of a Unix socket. Though it is not used for the TCP socket, it still needs to be a valid character array. `NULL` will not be accepted.

***port*** Number to identify socket. When a Unix socket is used, it becomes part of the socket name. When a TCP socket is used, it is used as the TCP port.

***data*** Data to associate with the created Ecore_Con_Server object.

**Returns:**

A new Ecore_Con_Server.

### 6.27.2.2 EAPI void ecore_con_server_client_limit_set (Ecore_Con_Server * *svr*, int *client_limit*, char *reject_excess_clients*)

Sets a limit on the number of clients that can be handled concurrently by the given server, and a policy on what to do if excess clients try to connect.

Beware that if you set this once ecore is already running, you may already have pending CLIENT_ADD events in your event queue. Those clients have already connected and will not be affected by this call. Only clients subsequently trying to connect will be affected.

**Parameters:**

***svr*** The given server.

***client_limit*** The maximum number of clients to handle concurrently. -1 means unlimited (default). 0 effectively disables the server.

***reject_excess_clients*** Set to 1 to automatically disconnect excess clients as soon as they connect if you are already handling client_limit clients. Set to 0 (default) to just hold off on the "accept()" system call until the number of active clients drops. This causes the kernel to queue up to 4096 connections (or your kernel's limit, whichever is lower).

#### 6.27.2.3 EAPI Ecore_List* ecore_con_server_clients_get (Ecore_Con_Server * *svr*)

Retrieves the current list of clients.

**Parameters:**

***svr*** The given server.

**Returns:**

The list of clients on this server.

#### 6.27.2.4 EAPI Ecore_Con_Server* ecore_con_server_connect (Ecore_Con_Type *compl_type*, const char * *name*, int *port*, const void * *data*)

Creates a server object to represent the server listening at the given port.

The socket to which the server connects depends on the connection type:

- If *compl_type* is `ECORE_CON_LOCAL_USER`, the function will connect to the server listening on the Unix socket "~/.ecore/[name]/[port]".
- If *compl_type* is `ECORE_CON_LOCAL_SYSTEM`, the function will connect to the server listening on the Unix socket "/tmp/.ecore_service|[name]|[port]".
- If *compl_type* is `ECORE_CON_REMOTE_SYSTEM`, the function will connect to the server listening on the TCP port "[name]:[port]".

**Parameters:**

***compl_type*** The connection type.

***name*** Name used when determining what socket to connect to. It is used to generate the socket name when the socket is a Unix socket. It is used as the hostname when connecting with a TCP socket.

***port*** Number to identify the socket to connect to. Used when generating the socket name for a Unix socket, or as the TCP port when connecting to a TCP socket.

***data*** Data to associate with the created Ecore_Con_Server object.

**Returns:**

A new Ecore_Con_Server.

#### 6.27.2.5 EAPI int ecore_con_server_connected_get (Ecore_Con_Server * *svr*)

Retrieves whether the given server is currently connected.

**Todo**

Check that this function does what the documenter believes it does.

**Parameters:**

*svr* The given server.

**Returns:**

1 if the server is connected. 0 otherwise.

#### 6.27.2.6 EAPI void* ecore_con_server_data_get (Ecore_Con_Server * svr)

Retrieves the data associated with the given server.

**Parameters:**

*svr* The given server.

**Returns:**

The associated data.

#### 6.27.2.7 EAPI void* ecore_con_server_del (Ecore_Con_Server * svr)

Closes the connection and frees the given server.

**Parameters:**

*svr* The given server.

**Returns:**

Data associated with the server when it was created.

#### 6.27.2.8 EAPI void ecore_con_server_flush (Ecore_Con_Server * svr)

Flushes all pending data to the given server.

Will return when done.

**Parameters:**

*svr* The given server.

#### 6.27.2.9 EAPI char* ecore_con_server_ip_get (Ecore_Con_Server * svr)

Gets the IP address of a server that has been connected to.

**Parameters:**

***svr*** The given server.

**Returns:**

A pointer to an internal string that contains the IP address of the connected server in the form "XXX.YYY.ZZZ.AAA" IP notation. This string should not be modified or trusted to stay valid after deletion for the `svr` object. If no IP is known NULL is returned.

#### 6.27.2.10 EAPI int ecore_con_server_send (Ecore_Con_Server * *svr*, const void * *data*, int *size*)

Sends the given data to the given server.

**Parameters:**

***svr*** The given server.

***data*** The given data.

***size*** Length of the data, in bytes, to send.

**Returns:**

The number of bytes sent. `0` will be returned if there is an error.

## 6.28 Ecore Connection Client Functions

Functions that operate on Ecore connection client objects.

### Functions

- EAPI int ecore_con_client_send (Ecore_Con_Client *cl, void *data, int size)

  *Sends the given data to the given client.*

- EAPI Ecore_Con_Server * ecore_con_client_server_get (Ecore_Con_Client *cl)

  *Retrieves the server representing the socket the client has connected to.*

- EAPI void * ecore_con_client_del (Ecore_Con_Client *cl)

  *Closes the connection and frees memory allocated to the given client.*

- EAPI void ecore_con_client_data_set (Ecore_Con_Client *cl, const void *data)

  *Sets the data associated with the given client to* `data`.

- EAPI void * ecore_con_client_data_get (Ecore_Con_Client *cl)

  *Retrieves the data associated with the given client.*

- EAPI char * ecore_con_client_ip_get (Ecore_Con_Client *cl)

  *Gets the IP address of a cleint that has connected.*

- EAPI void ecore_con_client_flush (Ecore_Con_Client *cl)

  *Flushes all pending data to the given client.*

- EAPI int ecore_con_ssl_available_get (void)

  *Returns if SSL support is available.*

- EAPI int ecore_ipc_ssl_available_get (void)

  *Returns if SSL support is available.*

### 6.28.1 Detailed Description

Functions that operate on Ecore connection client objects.

### 6.28.2 Function Documentation

#### 6.28.2.1 EAPI void* ecore_con_client_data_get (Ecore_Con_Client * *cl*)

Retrieves the data associated with the given client.

**Parameters:**

***cl*** The given client.

**Returns:**

The data associated with `cl`.

#### 6.28.2.2 EAPI void ecore_con_client_data_set (Ecore_Con_Client * *cl*, const void * *data*)

Sets the data associated with the given client to `data`.

**Parameters:**

***cl*** The given client.

***data*** What to set the data to.

#### 6.28.2.3 EAPI void* ecore_con_client_del (Ecore_Con_Client * *cl*)

Closes the connection and frees memory allocated to the given client.

**Parameters:**

***cl*** The given client.

**Returns:**

Data associated with the client.

#### 6.28.2.4 EAPI void ecore_con_client_flush (Ecore_Con_Client * *cl*)

Flushes all pending data to the given client.

Will return when done.

**Parameters:**

***cl*** The given client.

#### 6.28.2.5 EAPI char* ecore_con_client_ip_get (Ecore_Con_Client * *cl*)

Gets the IP address of a cleint that has connected.

**Parameters:**

***cl*** The given client.

**Returns:**

A pointer to an internal string that contains the IP address of the connected client in the form "XXX.YYY.ZZZ.AAA" IP notation. This string should not be modified or trusted to stay valid after deletion for the `cl` object. If no IP is known NULL is returned.

#### 6.28.2.6 EAPI int ecore_con_client_send (Ecore_Con_Client * *cl*, void * *data*, int *size*)

Sends the given data to the given client.

**Parameters:**

***cl*** The given client.

***data*** The given data.

***size*** Length of the data, in bytes, to send.

**Returns:**

The number of bytes sent. `0` will be returned if there is an error.

#### 6.28.2.7 EAPI Ecore_Con_Server* ecore_con_client_server_get (Ecore_Con_Client * *cl*)

Retrieves the server representing the socket the client has connected to.

**Parameters:**

***cl*** The given client.

**Returns:**

The server that the client connected to.

#### 6.28.2.8 EAPI int ecore_con_ssl_available_get (void)

Returns if SSL support is available.

**Returns:**

1 if SSL is available, 0 if it is not.

#### 6.28.2.9 EAPI int ecore_ipc_ssl_available_get (void)

Returns if SSL support is available.

**Returns:**

1 if SSL is available, 0 if it is not.

## 6.29 Ecore Config Property Functions

Functions that retrieve or set the attributes relating to a property.

### Functions

- EAPI Ecore_Config_Prop * ecore_config_dst (Ecore_Config_Prop *e)
  *Removes the given property from the local configuration and destroys it.*
- EAPI const char * ecore_config_type_get (const Ecore_Config_Prop *e)
  *Returns the type of the property.*
- EAPI int ecore_config_describe (const char *key, const char *desc)
  *Sets the description field of the indicated property.*
- EAPI int ecore_config_short_opt_set (const char *key, char short_opt)
  *Set the short option character of a property.*
- EAPI int ecore_config_long_opt_set (const char *key, const char *long_opt)
  *Set the long option string of the property.*
- EAPI int ecore_config_typed_set (const char *key, const void *val, int type)
  *Sets the indicated property to the given value and type.*

### 6.29.1 Detailed Description

Functions that retrieve or set the attributes relating to a property.

### 6.29.2 Function Documentation

#### 6.29.2.1 EAPI int ecore_config_describe (const char * *key*, const char * *desc*)

Sets the description field of the indicated property.

**Parameters:**

***key*** The property key.
***desc*** Description string.

**Note:**

The description string is copied for the property's use. You can free `desc` once this function is called.

#### 6.29.2.2 EAPI Ecore_Config_Prop* ecore_config_dst (Ecore_Config_Prop * e)

Removes the given property from the local configuration and destroys it.

**Parameters:**

***e*** Property to destroy.

**Returns:**

`NULL`

#### 6.29.2.3 EAPI int ecore_config_long_opt_set (const char * *key*, const char * *long_opt*)

Set the long option string of the property.

**Parameters:**

***key*** The property key.

***long_opt*** String used to indicate the value of a property given on the command line.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` on success. `ECORE_CONFIG_ERR_NODATA` is returned if the property does not exist.

#### 6.29.2.4 EAPI int ecore_config_short_opt_set (const char * *key*, char *short_opt*)

Set the short option character of a property.

**Parameters:**

***key*** The property key.

***short_opt*** Character used to indicate the value of a property given on the command line.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` on success. `ECORE_CONFIG_ERR_NODATA` is returned if the property does not exist.

#### 6.29.2.5 EAPI const char* ecore_config_type_get (const Ecore_Config_Prop * e)

Returns the type of the property.

**Parameters:**

***e*** Property to get the type of.

**Returns:**

The type of the property. If the property is invalid, then the string "not found" is returned.

#### 6.29.2.6 EAPI int ecore_config_typed_set (const char * *key*, const void * *val*, int *type*)

Sets the indicated property to the given value and type.

**Parameters:**

***key*** The property key.

***val*** A pointer to the value to set the property to.

***type*** The type of the property.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if the property is set successfully.

## 6.30 Configuration Retrieve Functions

Functions that retrieve configuration values, based on type.

### Functions

- EAPI Ecore_Config_Prop * ecore_config_get (const char *key)
  *Returns the property with the given key.*
- EAPI char * ecore_config_string_get (const char *key)
  *Returns the specified property as a string.*
- EAPI int ecore_config_boolean_get (const char *key)
  *Returns the specified property as an integer.*
- EAPI long ecore_config_int_get (const char *key)
  *Returns the specified property as a long integer.*
- EAPI float ecore_config_float_get (const char *key)
  *Returns the specified property as a float.*
- EAPI int ecore_config_argb_get (const char *key, int *a, int *r, int *g, int *b)
  *Finds the alpha, red, green and blue values of a color property.*
- EAPI long ecore_config_argbint_get (const char *key)
  *Returns a color property as a long.*
- EAPI char * ecore_config_argbstr_get (const char *key)
  *Returns a color property as a string of hexadecimal characters.*
- EAPI char * ecore_config_theme_get (const char *key)
  *Returns a theme property.*
- EAPI char * ecore_config_as_string_get (const char *key)
  *Retrieves the key as a string.*

### 6.30.1 Detailed Description

Functions that retrieve configuration values, based on type.

### 6.30.2 Function Documentation

#### 6.30.2.1 EAPI int ecore_config_argb_get (const char * *key*, int * *a*, int * *r*, int * *g*, int * *b*)

Finds the alpha, red, green and blue values of a color property.

**Parameters:**

***key*** The property key.

***a*** A pointer to an integer to store the alpha value into.

***r*** A pointer to an integer to store the red value into.

***g*** A pointer to an integer to store the green value into.

***b*** A pointer to an integer to store the blue value into.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` on success. `ECORE_CONFIG_ERR_FAIL` otherwise.

#### 6.30.2.2 EAPI long ecore_config_argbint_get (const char * *key*)

Returns a color property as a long.

**Parameters:**

***key*** The property key.

**Returns:**

ARGB data as long

#### 6.30.2.3 EAPI char* ecore_config_argbstr_get (const char * *key*)

Returns a color property as a string of hexadecimal characters.

**Parameters:**

***key*** The property key.

**Returns:**

A string of hexadecimal characters in the format aarrggbb.

#### 6.30.2.4 EAPI char* ecore_config_as_string_get (const char * *key*)

Retrieves the key as a string.

**Parameters:**

***key*** The property key.

**Returns:**

Returns a character array in the form of 'key:type=value'. `NULL` is returned if the property does not exist.

#### 6.30.2.5 EAPI int ecore_config_boolean_get (const char * *key*)

Returns the specified property as an integer.

**Parameters:**

***key*** The property key.

**Returns:**

The value of the property. The function returns -1 if the property is not an integer or is not set.

#### 6.30.2.6 EAPI float ecore_config_float_get (const char * *key*)

Returns the specified property as a float.

**Parameters:**

***key*** The property key.

**Returns:**

The float value of the property. The function returns 0.0 if the property is not a float or is not set.

#### 6.30.2.7 EAPI Ecore_Config_Prop* ecore_config_get (const char * *key*)

Returns the property with the given key.

**Parameters:**

***key*** The unique name of the wanted property.

**Returns:**

The property that corresponds to the given key. `NULL` if the key could not be found.

#### 6.30.2.8 EAPI long ecore_config_int_get (const char * *key*)

Returns the specified property as a long integer.

**Parameters:**

***key*** The property key.

**Returns:**

The integer value of the property. The function returns 0 if the property is not an integer or is not set.

#### 6.30.2.9 EAPI char* ecore_config_string_get (const char * *key*)

Returns the specified property as a string.

**Parameters:**

***key*** The property key.

**Returns:**

The string value of the property. The function returns `NULL` if the property is not a string or is not set.

#### 6.30.2.10 EAPI char* ecore_config_theme_get (const char * *key*)

Returns a theme property.

**Parameters:**

***key*** The property key.

**Returns:**

The name of the theme the property refers to. The function returns `NULL` if the property is not a theme or is not set.

## 6.31 Ecore Config Setters

Functions that set the value of a property.

### Functions

- EAPI int ecore_config_set (const char *key, const char *val)

  *Sets the indicated property to the value indicated by* val.

- EAPI int ecore_config_as_string_set (const char *key, const char *val)

  *Sets the indicated property to the value given in the string.*

- EAPI int ecore_config_boolean_set (const char *key, int val)

  *Sets the indicated property to the given boolean.*

- EAPI int ecore_config_int_set (const char *key, int val)

  *Sets the indicated property to the given integer.*

- EAPI int ecore_config_string_set (const char *key, const char *val)

  *Sets the indicated property to the given string.*

- EAPI int ecore_config_float_set (const char *key, float val)

  *Sets the indicated property to the given float value.*

- EAPI int ecore_config_argb_set (const char *key, int a, int r, int g, int b)

  *Sets the indicated property to a color value.*

- EAPI int ecore_config_argbint_set (const char *key, long argb)

  *Sets the indicated property to a color value.*

- EAPI int ecore_config_argbstr_set (const char *key, const char *val)

  *Sets the indicated property to a color value.*

- EAPI int ecore_config_theme_set (const char *key, const char *val)

  *Sets the indicated property to a theme name.*

- EAPI int ecore_config_theme_preview_group_set (const char *key, const char *group)

  *Sets the theme preview group of an indicated property.*

### 6.31.1 Detailed Description

Functions that set the value of a property.

### 6.31.2 Function Documentation

#### 6.31.2.1 EAPI int ecore_config_argb_set (const char * *key*, int *a*, int *r*, int *g*, int *b*)

Sets the indicated property to a color value.

**Parameters:**

***key*** The property key

***a*** integer 0..255

***r*** integer 0..255

***g*** integer 0..255

***b*** integer 0..255

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if the property is set successfully.

#### 6.31.2.2 EAPI int ecore_config_argbint_set (const char * *key*, long *argb*)

Sets the indicated property to a color value.

**Parameters:**

***key*** The property key

***argb*** ARGB data as long

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if the property is set successfully.

#### 6.31.2.3 EAPI int ecore_config_argbstr_set (const char * *key*, const char * *val*)

Sets the indicated property to a color value.

**Parameters:**

***key*** The property key

***val*** Color value in ARGB format.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if the property is set successfully.

#### 6.31.2.4 EAPI int ecore_config_as_string_set (const char * *key*, const char * *val*)

Sets the indicated property to the value given in the string.

**Parameters:**

***key*** The property key.

***val*** String representation of the value.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if the property is set successfully.

#### 6.31.2.5 EAPI int ecore_config_boolean_set (const char * *key*, int *val*)

Sets the indicated property to the given boolean.

**Parameters:**

***key*** The property key.

***val*** Boolean integer to set the property to.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if the property is set successfully.

#### 6.31.2.6 EAPI int ecore_config_float_set (const char * *key*, float *val*)

Sets the indicated property to the given float value.

**Parameters:**

***key*** The property key.

***val*** Float to set the property to.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if the property is set successfully.

#### 6.31.2.7 EAPI int ecore_config_int_set (const char * *key*, int *val*)

Sets the indicated property to the given integer.

**Parameters:**

***key*** The property key.

***val*** Integer to set the property to.

**Returns:**

ECORE_CONFIG_ERR_SUCC if the property is set successfully.

#### 6.31.2.8 EAPI int ecore_config_set (const char * *key*, const char * *val*)

Sets the indicated property to the value indicated by *val*.

**Parameters:**

***key*** The property key.

***val*** String representation of value to set.

**Returns:**

ECORE_CONFIG_ERR_SUCC if the property is set successfully.

#### 6.31.2.9 EAPI int ecore_config_string_set (const char * *key*, const char * *val*)

Sets the indicated property to the given string.

**Parameters:**

***key*** The property key.

***val*** String to set the property to.

**Returns:**

ECORE_CONFIG_ERR_SUCC if the property is set successfully.

#### 6.31.2.10 EAPI int ecore_config_theme_preview_group_set (const char * *key*, const char * *group*)

Sets the theme preview group of an indicated property.

**Parameters:**

***key*** The property key.

***group*** The group name.

**Returns:**

ECORE_CONFIG_ERR_SUCC on success.

#### 6.31.2.11 EAPI int ecore_config_theme_set (const char * *key*, const char * *val*)

Sets the indicated property to a theme name.

**Parameters:**

***key*** The property key.

***val*** String giving the name of the theme.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if the property is set successfully.

## 6.32 Ecore Config Defaults

Functions that are used to set the default values of properties.

### Functions

- EAPI int ecore_config_default (const char *key, const char *val, float lo, float hi, float step)

  *Sets the indicated property if it has not already been set or loaded.*

- EAPI int ecore_config_boolean_default (const char *key, int val)

  *Sets the indicated property to the given boolean if the property has not yet been set.*

- EAPI int ecore_config_int_default (const char *key, int val)

  *Sets the indicated property to the given integer if the property has not yet been set.*

- EAPI int ecore_config_int_default_bound (const char *key, int val, int low, int high, int step)

  *Sets the indicated property to the given integer if the property has not yet been set.*

- EAPI int ecore_config_string_default (const char *key, const char *val)

  *Sets the indicated property to the given string if the property has not yet been set.*

- EAPI int ecore_config_float_default (const char *key, float val)

  *Sets the indicated property to the given float if the property has not yet been set.*

- EAPI int ecore_config_float_default_bound (const char *key, float val, float low, float high, float step)

  *Sets the indicated property to the given float if the property has not yet been set.*

- EAPI int ecore_config_argb_default (const char *key, int a, int r, int g, int b)

  *Sets the indicated property to a color value if the property has not yet been set.*

- EAPI int ecore_config_argbint_default (const char *key, long argb)

  *Sets the indicated property to a color value if the property has not yet been set.*

- EAPI int ecore_config_argbstr_default (const char *key, const char *val)

  *Sets the indicated property to a color value if the property has not yet been set.*

- EAPI int ecore_config_theme_default (const char *key, const char *val)

  *Sets the indicated property to a theme name if the property has not yet been set.*

### 6.32.1 Detailed Description

Functions that are used to set the default values of properties.

### 6.32.2 Function Documentation

#### 6.32.2.1 EAPI int ecore_config_argb_default (const char * *key*, int *a*, int *r*, int *g*, int *b*)

Sets the indicated property to a color value if the property has not yet been set.

**Parameters:**

***key*** The property key.
***a*** integer 0..255
***r*** integer 0..255
***g*** integer 0..255
***b*** integer 0..255

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if there are no problems.

#### 6.32.2.2 EAPI int ecore_config_argbint_default (const char * *key*, long *argb*)

Sets the indicated property to a color value if the property has not yet been set.

**Parameters:**

***key*** The property key.
***argb*** ARGB data as long

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if there are no problems.

#### 6.32.2.3 EAPI int ecore_config_argbstr_default (const char * *key*, const char * *val*)

Sets the indicated property to a color value if the property has not yet been set.

**Parameters:**

***key*** The property key.
***val*** Color value in ARGB format.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if there are no problems.

#### 6.32.2.4 EAPI int ecore_config_boolean_default (const char * *key*, int *val*)

Sets the indicated property to the given boolean if the property has not yet been set.

**Parameters:**

***key*** The property key.

***val*** Boolean Integer to set the value to.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if there are no problems.

#### 6.32.2.5 EAPI int ecore_config_default (const char * *key*, const char * *val*, float *lo*, float *hi*, float *step*)

Sets the indicated property if it has not already been set or loaded.

**Parameters:**

***key*** The property key.

***val*** Default value of the key.

***lo*** Lowest valid value for the key.

***hi*** Highest valid value for the key.

***step*** Used by integer and float values.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if there are no errors.

**Note:**

The `lo`, `hi` and `step` parameters are only used when storing integer and float properties.

#### 6.32.2.6 EAPI int ecore_config_float_default (const char * *key*, float *val*)

Sets the indicated property to the given float if the property has not yet been set.

**Parameters:**

***key*** The property key.

***val*** Float to set the property to.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if there were no problems.

#### 6.32.2.7 EAPI int ecore_config_float_default_bound (const char * *key*, float *val*, float *low*, float *high*, float *step*)

Sets the indicated property to the given float if the property has not yet been set.

The bounds and step values are set regardless.

**Parameters:**

***key*** The property key.

***val*** Float to set the property to.

***low*** Lowest valid integer value for the property.

***high*** Highest valid float value for the property.

***step*** Increment value for the property.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if there were no problems.

#### 6.32.2.8 EAPI int ecore_config_int_default (const char * *key*, int *val*)

Sets the indicated property to the given integer if the property has not yet been set.

**Parameters:**

***key*** The property key.

***val*** Integer to set the value to.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if there are no problems.

#### 6.32.2.9 EAPI int ecore_config_int_default_bound (const char * *key*, int *val*, int *low*, int *high*, int *step*)

Sets the indicated property to the given integer if the property has not yet been set.

The bounds and step values are set regardless.

**Parameters:**

***key*** The property key.

***val*** Integer to set the property to.

***low*** Lowest valid integer value for the property.

***high*** Highest valid integer value for the property.

***step*** Increment value for the property.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if there were no problems.

#### 6.32.2.10 EAPI int ecore_config_string_default (const char * *key*, const char * *val*)

Sets the indicated property to the given string if the property has not yet been set.

**Parameters:**

***key*** The property key.

***val*** String to set the property to.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if there were no problems.

#### 6.32.2.11 EAPI int ecore_config_theme_default (const char * *key*, const char * *val*)

Sets the indicated property to a theme name if the property has not yet been set.

**Parameters:**

***key*** The property key.

***val*** String giving the name of the theme.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if the property is set successfully.

## 6.33 Ecore Config Structures

Functions that are used to create structures of properties.

### Functions

- EAPI int ecore_config_struct_create (const char *key)

  *Sets the indicated property to a structure if the property has not yet been set.*

- EAPI int ecore_config_struct_int_add (const char *key, const char *name, int val)

  *Add an int property to the named structure.*

- EAPI int ecore_config_struct_float_add (const char *key, const char *name, float val)

  *Add a float property to the named structure.*

- EAPI int ecore_config_struct_string_add (const char *key, const char *name, const char *val)

  *Add a string property to the named structure.*

- EAPI int ecore_config_struct_argb_add (const char *key, const char *name, int a, int r, int g, int b)

  *Add an argb property to the named structure.*

- EAPI int ecore_config_struct_theme_add (const char *key, const char *name, const char *val)

  *Add a theme property to the named structure.*

- EAPI int ecore_config_struct_boolean_add (const char *key, const char *name, int val)

  *Add a boolean property to the named structure.*

- EAPI int ecore_config_struct_get (const char *key, void *data)

  *Get the contents of a defined structure property and load it into the passed C struct.*

### 6.33.1 Detailed Description

Functions that are used to create structures of properties.

### 6.33.2 Function Documentation

#### 6.33.2.1 EAPI int ecore_config_struct_argb_add (const char * *key*, const char * *name*, int *a*, int *r*, int *g*, int *b*)

Add an argb property to the named structure.

The property is set if it has not yet been set.

**Parameters:**

***key*** The key of the structure to add to.

***name*** The name of the item to add - this will be appended to the key

***a*** The alpha to default to

***r*** The red to default to

***g*** The green to default to

***b*** The blue to default to

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if the property is set successfully.

#### 6.33.2.2 EAPI int ecore_config_struct_boolean_add (const char * *key*, const char * *name*, int *val*)

Add a boolean property to the named structure.

The property is set if it has not yet been set.

**Parameters:**

***key*** The key of the structure to add to.

***name*** The name of the item to add - this will be appended to the key

***val*** The boolean to default to

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if the property is set successfully.

#### 6.33.2.3 EAPI int ecore_config_struct_create (const char * *key*)

Sets the indicated property to a structure if the property has not yet been set.

**Parameters:**

***key*** The property key.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if the property is set successfully.

#### 6.33.2.4 EAPI int ecore_config_struct_float_add (const char * *key*, const char * *name*, float *val*)

Add a float property to the named structure.

The property is set if it has not yet been set.

**Parameters:**

***key*** The key of the structure to add to.

***name*** The name of the item to add - this will be appended to the key

***val*** The float to default to

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if the property is set successfully.

#### 6.33.2.5 EAPI int ecore_config_struct_get (const char * *key*, void * *data*)

Get the contents of a defined structure property and load it into the passed C struct.

**Parameters:**

***key*** The name of the structure property to look up.

***data*** The struct to write into.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if the structure is written successfully.

#### 6.33.2.6 EAPI int ecore_config_struct_int_add (const char * *key*, const char * *name*, int *val*)

Add an int property to the named structure.

The property is set if it has not yet been set.

**Parameters:**

***key*** The key of the structure to add to.

***name*** The name of the item to add - this will be appended to the key

***val*** the int to default to

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if the property is set successfully.

#### 6.33.2.7 EAPI int ecore_config_struct_string_add (const char * *key*, const char * *name*, const char * *val*)

Add a string property to the named structure.

The property is set if it has not yet been set.

**Parameters:**

***key*** The key of the structure to add to.

***name*** The name of the item to add - this will be appended to the key

***val*** The string to default to

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if the property is set successfully.

#### 6.33.2.8 EAPI int ecore_config_struct_theme_add (const char * *key*, const char * *name*, const char * *val*)

Add a theme property to the named structure.

The property is set if it has not yet been set.

**Parameters:**

***key*** The key of the structure to add to.

***name*** The name of the item to add - this will be appended to the key

***val*** The theme name to default to

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if the property is set successfully.

## 6.34 Ecore Config Listeners

Functions that set and unset property listener callbacks.

### Functions

- EAPI int ecore_config_listen (const char *name, const char *key, Ecore_Config_Listener listener, int tag, void *data)

  *Adds a callback function to the list of functions called when a property changes.*

- EAPI int ecore_config_deaf (const char *name, const char *key, Ecore_Config_Listener listener)

  *Removes a listener callback.*

### 6.34.1 Detailed Description

Functions that set and unset property listener callbacks.

### 6.34.2 Function Documentation

#### 6.34.2.1 EAPI int ecore_config_deaf (const char * *name*, const char * *key*, Ecore_Config_Listener *listener*)

Removes a listener callback.

**Parameters:**

***name*** Name of the callback to remove.

***key*** The property key the callback is listening to.

***listener*** The callback function to remove.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if successful in removing the callback. If no callback matches the given parameters, then `ECORE_CONFIG_ERR_NOTFOUND` is returned. If `NULL` is passed for the key pointer, `ECORE_CONFIG_ERR_NODATA` is returned.

#### 6.34.2.2 EAPI int ecore_config_listen (const char * *name*, const char * *key*, Ecore_Config_Listener *listener*, int *tag*, void * *data*)

Adds a callback function to the list of functions called when a property changes.

**Parameters:**

***name*** Name of the callback.

***key*** The key of the property to listen to.

***listener*** Listener callback function.

***tag*** Tag to pass to `listener` when it is called.

***data*** Data to pass to `listener` when it is called.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if successful in setting up the callback.

## 6.35 Ecore Config App Library Functions

Functions that are used to start up and shutdown the Enlightened Property Library when used directly by an application.

### Functions

- EAPI int ecore_config_init (const char *name)
  *Initializes the Enlightened Property Library.*
- EAPI int ecore_config_shutdown (void)
  *Frees memory and shuts down the library for an application.*

### 6.35.1 Detailed Description

Functions that are used to start up and shutdown the Enlightened Property Library when used directly by an application.

### 6.35.2 Function Documentation

#### 6.35.2.1 EAPI int ecore_config_init (const char * *name*)

Initializes the Enlightened Property Library.

Either this function or ecore_config_system_init must be run before any other function in the Enlightened Property Library, even if you have run ecore_init . The name given is used to determine the default configuration to load.

**Parameters:**

***name*** Application name

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if the library is successfully set up. `ECORE_CONFIG_ERR_FAIL` otherwise.

#### 6.35.2.2 EAPI int ecore_config_shutdown (void)

Frees memory and shuts down the library for an application.

**Returns:**

`ECORE_CONFIG_ERR_IGNORED` .

## 6.36 Ecore Config Library Functions

Functions that are used to start up and shutdown the Enlightened Property Library when used directly by an application.

### Functions

- EAPI int ecore_config_system_init (void)

  *Initializes the Enlightened Property Library.*

- EAPI int ecore_config_system_shutdown (void)

  *Frees memory and shuts down the library for other programming libraries.*

### 6.36.1 Detailed Description

Functions that are used to start up and shutdown the Enlightened Property Library when used directly by an application.

### 6.36.2 Function Documentation

#### 6.36.2.1 EAPI int ecore_config_system_init (void)

Initializes the Enlightened Property Library.

This function is meant to be run from other programming libraries. It should not be called from applications.

This function (or ecore_config_init ) must be run before any other function in the Enlightened Property Library, even if you have run ecore_init .

**Returns:**

`ECORE_CONFIG_ERR_SUCC` if the library is successfully set up. `ECORE_CONFIG_ERR_FAIL` otherwise.

#### 6.36.2.2 EAPI int ecore_config_system_shutdown (void)

Frees memory and shuts down the library for other programming libraries.

**Returns:**

`ECORE_CONFIG_ERR_IGNORED`

## 6.37 .desktop file Functions

Functions that deal with freedesktop.org desktop files.

### Functions

- Ecore_Hash * ecore_desktop_ini_get (const char *file)
  *Get the contents of a .ini style file.*
- Ecore_Desktop * ecore_desktop_get (const char *file, const char *lang)
  *Get the contents of a .desktop file.*
- EAPI int ecore_desktop_init ()
  *Setup what ever needs to be setup to support Ecore_Desktop.*
- EAPI int ecore_desktop_shutdown ()
  *Tear down what ever needs to be torn down to support Ecore_Desktop.*
- void ecore_desktop_destroy (Ecore_Desktop *desktop)
  *Free whatever resources are used by an Ecore_Desktop.*
- char * ecore_desktop_home_get ()
  *Get and massage the users home directory.*

### 6.37.1 Detailed Description

Functions that deal with freedesktop.org desktop files.

This conforms with the freedesktop.org XDG Desktop Entry Specification version 0.9.4

### 6.37.2 Function Documentation

#### 6.37.2.1 void ecore_desktop_destroy (Ecore_Desktop * *desktop*)

Free whatever resources are used by an Ecore_Desktop.

There are internal resources used by each Ecore_Desktop This releases those resources.

**Parameters:**

***desktop*** An Ecore_Desktop that was previously returned by ecore_desktop_get().

#### 6.37.2.2 Ecore_Desktop* ecore_desktop_get (const char * *file*, const char * *lang*)

Get the contents of a .desktop file.

Use ecore_desktop_destroy() to free this structure.

**Parameters:**

***file*** Full path to the .desktop file.

***lang*** Language to use, or NULL for default.

**Returns:**

An Ecore_Desktop containing the files contents.

#### 6.37.2.3 char* ecore_desktop_home_get (void)

Get and massage the users home directory.

This is an internal function that may be useful elsewhere.

**Returns:**

The users howe directory.

#### 6.37.2.4 Ecore_Hash* ecore_desktop_ini_get (const char * *file*)

Get the contents of a .ini style file.

The Ecore_Hash returned is a two level hash, the first level is the groups in the file, one per group, keyed by the name of that group. The value of each of those first level hashes is the second level Ecore_Hash, the contents of each group.

**Parameters:**

***file*** Full path to the .ini style file.

**Returns:**

An Ecore_Hash of the files contents.

#### 6.37.2.5 EAPI int ecore_desktop_init (void)

Setup what ever needs to be setup to support Ecore_Desktop.

There are internal structures that are needed for Ecore_Desktop functions to operate, this sets them up.

#### 6.37.2.6 EAPI int ecore_desktop_shutdown (void)

Tear down what ever needs to be torn down to support Ecore_Desktop.

There are internal structures that are needed for Ecore_Desktop functions to operate, this tears them down.

## 6.38 icon theme Functions

Functions that deal with freedesktop.org icon themes.

### Functions

- EAPI char * ecore_desktop_icon_find (const char *icon, const char *icon_size, const char *icon_theme)

  *Find the path to an icon.*

- EAPI int ecore_desktop_icon_init ()

  *Setup what ever needs to be setup to support ecore_desktop_icon.*

- EAPI int ecore_desktop_icon_shutdown ()

  *Tear down what ever needs to be torn down to support ecore_desktop_ycon.*

- Ecore_Desktop_Icon_Theme * ecore_desktop_icon_theme_get (const char *icon_theme, const char *lang __UNUSED__)

  *Get the contents of an index.theme file.*

- void ecore_desktop_icon_theme_destroy (Ecore_Desktop_Icon_Theme *icon_theme)

  *Free whatever resources are used by an Ecore_Desktop_Icon_Theme.*

### 6.38.1 Detailed Description

Functions that deal with freedesktop.org icon themes.

This conforms with the freedesktop.org XDG Icon Theme Specification version 0.11

### 6.38.2 Function Documentation

#### 6.38.2.1 EAPI char* ecore_desktop_icon_find (const char * *icon*, const char * *icon_size*, const char * *icon_theme*)

Find the path to an icon.

Using the search algorithm specified by freedesktop.org, search for an icon in the currently installed set of icon themes.

The returned string needs to be freed eventually.

**Parameters:**

***icon*** The name of the required icon.

***icon_size*** The size of the required icon.

***icon_theme*** The theme of the required icon.

**Returns:**

The full path to an icon file, or NULL.

#### 6.38.2.2 EAPI int ecore_desktop_icon_init (void)

Setup what ever needs to be setup to support ecore_desktop_icon.

There are internal structures that are needed for ecore_desktop_icon functions to operate, this sets them up.

#### 6.38.2.3 EAPI int ecore_desktop_icon_shutdown (void)

Tear down what ever needs to be torn down to support ecore_desktop_ycon.

There are internal structures that are needed for ecore_desktop_icon functions to operate, this tears them down.

#### 6.38.2.4 void ecore_desktop_icon_theme_destroy (Ecore_Desktop_Icon_Theme * *icon_theme*)

Free whatever resources are used by an Ecore_Desktop_Icon_Theme.

There are internal resources used by each Ecore_Desktop_Icon_Theme This releases those resources.

**Parameters:**

***icon_theme*** An Ecore_Desktop_Icon_Theme.

#### 6.38.2.5 Ecore_Desktop_Icon_Theme* ecore_desktop_icon_theme_get (const char * *icon_theme*, const char *lang __ *UNUSED* __)

Get the contents of an index.theme file.

Everything that is in the index.theme file is returned in the data member of the Ecore_Desktop_Icon_Theme structure, it's an Ecore_Hash as returned by ecore_desktop_ini_get(). Some of the data in the index.theme file is decoded into specific members of the returned structure.

Use ecore_desktop_icon_theme_destroy() to free this structure.

**Parameters:**

***icon_theme*** Name of the icon theme, or full path to the index.theme file.

***lang*** Language to use, or NULL for default.

**Returns:**

An Ecore_Desktop_Icon_Theme containing the files contents.

## 6.39 menu Functions

Functions that deal with freedesktop.org menus.

### Functions

- Ecore_Desktop_Tree * ecore_desktop_menu_get (char *file)
  *Decode a freedesktop.org menu XML jungle.*

### 6.39.1 Detailed Description

Functions that deal with freedesktop.org menus.

### 6.39.2 Function Documentation

#### 6.39.2.1 Ecore_Desktop_Tree* ecore_desktop_menu_get (char * *file*)

Decode a freedesktop.org menu XML jungle.

Using the algorithm specified by freedesktop.org, fully decode a menu based on an initial menu file.

**Parameters:**

***file*** The base file for the menu.

**Returns:**

The resulting menu tree.

## 6.40 Framebuffer Library Functions

Functions used to set up and shut down the Ecore_Framebuffer functions.

### Functions

- EAPI int ecore_fb_init (const char *name __UNUSED__)
  *Sets up the Ecore_Fb library.*
- EAPI int ecore_fb_shutdown (void)
  *Shuts down the Ecore_Fb library.*

### 6.40.1 Detailed Description

Functions used to set up and shut down the Ecore_Framebuffer functions.

### 6.40.2 Function Documentation

#### 6.40.2.1 EAPI int ecore_fb_init (const char *name __ *UNUSED*__ )

Sets up the Ecore_Fb library.

**Parameters:**

***name*** device target name

**Returns:**

`0` on failure. Otherwise, the number of times the library has been initialised without being shut down.

#### 6.40.2.2 EAPI int ecore_fb_shutdown (void)

Shuts down the Ecore_Fb library.

**Returns:**

`The` number of times the system has been initialised without being shut down.

## 6.41 Framebuffer Double Click Functions

Functions that deal with the double click time of the framebuffer.

### Functions

- EAPI void ecore_fb_double_click_time_set (double t)
  *Sets the timeout for a double and triple clicks to be flagged.*
- EAPI double ecore_fb_double_click_time_get (void)
  *Retrieves the double and triple click flag timeout.*

### 6.41.1 Detailed Description

Functions that deal with the double click time of the framebuffer.

### 6.41.2 Function Documentation

#### 6.41.2.1 EAPI double ecore_fb_double_click_time_get (void)

Retrieves the double and triple click flag timeout.

See ecore_x_double_click_time_set for more information.

**Returns:**

The timeout for double clicks in seconds.

#### 6.41.2.2 EAPI void ecore_fb_double_click_time_set (double *t*)

Sets the timeout for a double and triple clicks to be flagged.

This sets the time between clicks before the double_click flag is set in a button down event. If 3 clicks occur within double this time, the triple_click flag is also set.

**Parameters:**

***t*** The time in seconds

## 6.42 Framebuffer Calibration Functions

Functions that calibrate the screen.

### Functions

- EAPI void ecore_fb_touch_screen_calibrate_set (int xscale, int xtrans, int yscale, int ytrans, int xyswap)

  *Calibrates the touschreen using the given parameters.*

- EAPI void ecore_fb_touch_screen_calibrate_get (int *xscale, int *xtrans, int *yscale, int *ytrans, int *xyswap)

  *Retrieves the calibration parameters of the touchscreen.*

### 6.42.1 Detailed Description

Functions that calibrate the screen.

### 6.42.2 Function Documentation

#### 6.42.2.1 EAPI void ecore_fb_touch_screen_calibrate_get (int * *xscale*, int * *xtrans*, int * *yscale*, int * *ytrans*, int * *xyswap*)

Retrieves the calibration parameters of the touchscreen.

**Parameters:**

***xscale*** Pointer to an integer in which to store the X scaling. Note that 256 = 1.0.

***xtrans*** Pointer to an integer in which to store the X translation.

***yscale*** Pointer to an integer in which to store the Y scaling.

***ytrans*** Pointer to an integer in which to store the Y translation.

***xyswap*** Pointer to an integer in which to store the Swap X & Y flag.

#### 6.42.2.2 EAPI void ecore_fb_touch_screen_calibrate_set (int *xscale*, int *xtrans*, int *yscale*, int *ytrans*, int *xyswap*)

Calibrates the touschreen using the given parameters.

**Parameters:**

***xscale*** X scaling, where 256 = 1.0

***xtrans*** X translation.

***yscale*** Y scaling.

***ytrans*** Y translation.

***xyswap*** Swap X & Y flag.

## 6.43 Framebuffer Backlight Functions

Functions that deal with the backlight of a framebuffer's screen.

### Functions

- EAPI void ecore_fb_backlight_set (int on)
  *Turns on or off the backlight.*
- EAPI int ecore_fb_backlight_get (void)
  *Retrieves the backlight state.*
- EAPI void ecore_fb_backlight_brightness_set (double br)
  *Sets the backlight brightness.*
- EAPI double ecore_fb_backlight_brightness_get (void)
  *Retrieves the backlight brightness.*

### 6.43.1 Detailed Description

Functions that deal with the backlight of a framebuffer's screen.

### 6.43.2 Function Documentation

#### 6.43.2.1 EAPI double ecore_fb_backlight_brightness_get (void)

Retrieves the backlight brightness.

**Returns:**

The current backlight brigntess, where 0.0 is the darkest and 1.0 is the brightest.

#### 6.43.2.2 EAPI void ecore_fb_backlight_brightness_set (double *br*)

Sets the backlight brightness.

**Parameters:**

***br*** Brightness between 0.0 to 1.0, where 0.0 is darkest and 1.0 is brightest.

#### 6.43.2.3 EAPI int ecore_fb_backlight_get (void)

Retrieves the backlight state.

**Returns:**

Whether the backlight is on.

#### 6.43.2.4 EAPI void ecore_fb_backlight_set (int *on*)

Turns on or off the backlight.

**Parameters:**

***on*** `1` to turn the backlight on. `0` to turn it off.

## 6.44 Framebuffer LED Functions

Functions that deal with the light emitting diode connected to the current framebuffer.

### Functions

- EAPI void ecore_fb_led_set (int on)
  *Sets whether the current framebuffer's LED to the given state.*
- EAPI void ecore_fb_led_blink_set (double speed)
  *Makes the LED of the current framebuffer blink.*

### 6.44.1 Detailed Description

Functions that deal with the light emitting diode connected to the current framebuffer.

### 6.44.2 Function Documentation

#### 6.44.2.1 EAPI void ecore_fb_led_blink_set (double *speed*)

Makes the LED of the current framebuffer blink.

**Parameters:**

***speed*** Number to give the speed on the blink.

**Todo**

Documentation: Work out what speed the units are in.

#### 6.44.2.2 EAPI void ecore_fb_led_set (int *on*)

Sets whether the current framebuffer's LED to the given state.

**Parameters:**

***on*** `1` to indicate the LED should be on, `0` if it should be off.

## 6.45 Framebuffer Contrast Functions

Values that set and retrieve the contrast of a framebuffer screen.

### Functions

- EAPI void ecore_fb_contrast_set (double cr)
  *Sets the contrast used by the framebuffer screen.*
- EAPI double ecore_fb_contrast_get (void)
  *Retrieves the contrast currently being used by the framebuffer screen.*

### 6.45.1 Detailed Description

Values that set and retrieve the contrast of a framebuffer screen.

### 6.45.2 Function Documentation

#### 6.45.2.1 EAPI double ecore_fb_contrast_get (void)

Retrieves the contrast currently being used by the framebuffer screen.

**Returns:**

A value between 0 and 1 that represents the current contrast of the screen.

#### 6.45.2.2 EAPI void ecore_fb_contrast_set (double *cr*)

Sets the contrast used by the framebuffer screen.

**Parameters:**

***cr*** Value between 0 and 1 that gives the new contrast of the screen.

## 6.46 IPC Library Functions

Functions that set up and shut down the Ecore IPC Library.

### Functions

- EAPI int ecore_ipc_init (void)
  *Initialises the Ecore IPC library.*
- EAPI int ecore_ipc_shutdown (void)
  *Shuts down the Ecore IPC library.*

### 6.46.1 Detailed Description

Functions that set up and shut down the Ecore IPC Library.

### 6.46.2 Function Documentation

#### 6.46.2.1 EAPI int ecore_ipc_init (void)

Initialises the Ecore IPC library.

**Returns:**

Number of times the library has been initialised without being shut down.

#### 6.46.2.2 EAPI int ecore_ipc_shutdown (void)

Shuts down the Ecore IPC library.

**Returns:**

Number of times the library has been initialised without being shut down.

## 6.47 IPC Server Functions

Functions the deal with IPC server objects.

### Functions

- EAPI Ecore_Ipc_Server * ecore_ipc_server_add (Ecore_Ipc_Type compl_type, const char *name, int port, const void *data)

  *Creates an IPC server that listens for connections.*

- EAPI Ecore_Ipc_Server * ecore_ipc_server_connect (Ecore_Ipc_Type compl_type, char *name, int port, const void *data)

  *Creates an IPC server object to represent the IPC server listening on the given port.*

- EAPI void * ecore_ipc_server_del (Ecore_Ipc_Server *svr)

  *Closes the connection and frees the given IPC server.*

- EAPI void * ecore_ipc_server_data_get (Ecore_Ipc_Server *svr)

  *Retrieves the data associated with the given IPC server.*

- EAPI int ecore_ipc_server_connected_get (Ecore_Ipc_Server *svr)

  *Retrieves whether the given IPC server is currently connected.*

- EAPI Ecore_List * ecore_ipc_server_clients_get (Ecore_Ipc_Server *svr)

  *Retrieves the list of clients for this server.*

- EAPI int ecore_ipc_server_send (Ecore_Ipc_Server *svr, int major, int minor, int ref, int ref_to, int response, void *data, int size)

  *Sends a message to the given IPC server.*

### 6.47.1 Detailed Description

Functions the deal with IPC server objects.

### 6.47.2 Function Documentation

#### 6.47.2.1 EAPI Ecore_Ipc_Server* ecore_ipc_server_add (Ecore_Ipc_Type *compl_type*, const char * *name*, int *port*, const void * *data*)

Creates an IPC server that listens for connections.

For more details about the `compl_type`, `name` and `port` parameters, see the ecore_con_server_add documentation.

**Parameters:**

***compl_type*** The connection type.

***name*** Name to associate with the socket used for connection.

***port*** Number to identify with socket used for connection.

***data*** Data to associate with the IPC server.

**Returns:**

New IPC server. If there is an error, `NULL` is returned.

**Todo**

Need to add protocol type parameter to this function.

#### 6.47.2.2 EAPI Ecore_List* ecore_ipc_server_clients_get (Ecore_Ipc_Server * *svr*)

Retrieves the list of clients for this server.

**Parameters:**

***svr*** The given IPC server.

**Returns:**

An Ecore_List with the clients.

#### 6.47.2.3 EAPI Ecore_Ipc_Server* ecore_ipc_server_connect (Ecore_Ipc_Type *compl_type*, char * *name*, int *port*, const void * *data*)

Creates an IPC server object to represent the IPC server listening on the given port.

For more details about the `compl_type`, `name` and `port` parameters, see the ecore_con_server_connect documentation.

**Parameters:**

***compl_type*** The IPC connection type.

***name*** Name used to determine which socket to use for the IPC connection.

***port*** Number used to identify the socket to use for the IPC connection.

***data*** Data to associate with the server.

**Returns:**

A new IPC server. `NULL` is returned on error.

**Todo**

Need to add protocol type parameter.

#### 6.47.2.4 EAPI int ecore_ipc_server_connected_get (Ecore_Ipc_Server * *svr*)

Retrieves whether the given IPC server is currently connected.

**Parameters:**

***svr*** The given IPC server.

**Returns:**

`1` if the server is connected. `0` otherwise.

#### 6.47.2.5 EAPI void* ecore_ipc_server_data_get (Ecore_Ipc_Server * *svr*)

Retrieves the data associated with the given IPC server.

**Parameters:**

***svr*** The given IPC server.

**Returns:**

The associated data.

#### 6.47.2.6 EAPI void* ecore_ipc_server_del (Ecore_Ipc_Server * *svr*)

Closes the connection and frees the given IPC server.

**Parameters:**

***svr*** The given IPC server.

**Returns:**

The data associated with the server when it was created.

#### 6.47.2.7 EAPI int ecore_ipc_server_send (Ecore_Ipc_Server * *svr*, int *major*, int *minor*, int *ref*, int *ref_to*, int *response*, void * *data*, int *size*)

Sends a message to the given IPC server.

The content of the parameters, excluding the `svr` paramter, is up to the client.

**Parameters:**

***svr*** The given IPC server.

***major*** Major opcode of the message.

***minor*** Minor opcode of the message.

***ref*** Message reference number.

***ref_to*** Reference number of the message this message refers to.

***response*** Requires response.

***data*** The data to send as part of the message.

***size*** Length of the data, in bytes, to send.

**Returns:**

Number of bytes sent. `0` is returned if there is an error.

**Todo**

This function needs to become an IPC message.

**Todo**

Fix up the documentation: Make sure what ref_to and response are.

## 6.48 IPC Client Functions

Functions that deal with IPC client objects.

### Functions

- EAPI int ecore_ipc_client_send (Ecore_Ipc_Client *cl, int major, int minor, int ref, int ref_to, int response, void *data, int size)

  *Sends a message to the given IPC client.*

- EAPI Ecore_Ipc_Server * ecore_ipc_client_server_get (Ecore_Ipc_Client *cl)

  *Retrieves the IPC server that the given IPC client is connected to.*

- EAPI void * ecore_ipc_client_del (Ecore_Ipc_Client *cl)

  *Closes the connection and frees memory allocated to the given IPC client.*

- EAPI void ecore_ipc_client_data_set (Ecore_Ipc_Client *cl, const void *data)

  *Sets the IPC data associated with the given IPC client to* `data`.

- EAPI void * ecore_ipc_client_data_get (Ecore_Ipc_Client *cl)

  *Retrieves the data that has been associated with the given IPC client.*

### 6.48.1 Detailed Description

Functions that deal with IPC client objects.

### 6.48.2 Function Documentation

#### 6.48.2.1 EAPI void* ecore_ipc_client_data_get (Ecore_Ipc_Client * *cl*)

Retrieves the data that has been associated with the given IPC client.

**Parameters:**

***cl*** The given client.

**Returns:**

The data associated with the IPC client.

#### 6.48.2.2 EAPI void ecore_ipc_client_data_set (Ecore_Ipc_Client * *cl*, const void * *data*)

Sets the IPC data associated with the given IPC client to `data`.

**Parameters:**

***cl*** The given IPC client.

***data*** The data to associate with the IPC client.

#### 6.48.2.3 EAPI void* ecore_ipc_client_del (Ecore_Ipc_Client * *cl*)

Closes the connection and frees memory allocated to the given IPC client.

**Parameters:**

***cl*** The given client.

**Returns:**

Data associated with the client.

#### 6.48.2.4 EAPI int ecore_ipc_client_send (Ecore_Ipc_Client * *cl*, int *major*, int *minor*, int *ref*, int *ref_to*, int *response*, void * *data*, int *size*)

Sends a message to the given IPC client.

**Parameters:**

***cl*** The given IPC client.

***major*** Major opcode of the message.

***minor*** Minor opcode of the message.

***ref*** Reference number of the message.

***ref_to*** Reference number of the message this message refers to.

***response*** Requires response.

***data*** The data to send as part of the message.

***size*** Length of the data, in bytes, to send.

**Returns:**

The number of bytes sent. `0` will be returned if there is an error.

**Todo**

This function needs to become an IPC message.

**Todo**

Make sure ref_to and response parameters are described correctly.

#### 6.48.2.5 EAPI Ecore_Ipc_Server* ecore_ipc_client_server_get (Ecore_Ipc_Client * *cl*)

Retrieves the IPC server that the given IPC client is connected to.

**Parameters:**

***cl*** The given IPC client.

**Returns:**

The IPC server the IPC client is connected to.

## 6.49 X Library Init and Shutdown Functions

Functions that start and shut down the Ecore X Library.

### Functions

- EAPI int ecore_x_init (const char *name)

  *Initialize the X display connection to the given display.*

- EAPI int ecore_x_shutdown (void)

  *Shuts down the Ecore X library.*

- EAPI int ecore_x_disconnect (void)

  *Shuts down the Ecore X library.*

### 6.49.1 Detailed Description

Functions that start and shut down the Ecore X Library.

### 6.49.2 Function Documentation

#### 6.49.2.1 EAPI int ecore_x_disconnect (void)

Shuts down the Ecore X library.

As ecore_x_shutdown, except do not close Display, only connection.

#### 6.49.2.2 EAPI int ecore_x_init (const char * *name*)

Initialize the X display connection to the given display.

**Parameters:**

***name*** Display target name. If `NULL`, the default display is assumed.

**Returns:**

The number of times the library has been initialized without being shut down. 0 is returned if an error occurs.

#### 6.49.2.3 EAPI int ecore_x_shutdown (void)

Shuts down the Ecore X library.

In shutting down the library, the X display connection is terminated and any event handlers for it are removed.

**Returns:**

The number of times the library has been initialized without being shut down.

## 6.50 X Display Attributes

Functions that set and retrieve X display attributes.

### Functions

- EAPI Ecore_X_Display * ecore_x_display_get (void)
  *Retrieves the Ecore_X_Display handle used for the current X connection.*
- EAPI int ecore_x_fd_get (void)
  *Retrieves the X display file descriptor.*
- EAPI void ecore_x_double_click_time_set (double t)
  *Sets the timeout for a double and triple clicks to be flagged.*
- EAPI double ecore_x_double_click_time_get (void)
  *Retrieves the double and triple click flag timeout.*

### 6.50.1 Detailed Description

Functions that set and retrieve X display attributes.

### 6.50.2 Function Documentation

#### 6.50.2.1 EAPI Ecore_X_Display* ecore_x_display_get (void)

Retrieves the Ecore_X_Display handle used for the current X connection.

**Returns:**

The current X display.

#### 6.50.2.2 EAPI double ecore_x_double_click_time_get (void)

Retrieves the double and triple click flag timeout.

See ecore_x_double_click_time_set for more information.

**Returns:**

The timeout for double clicks in seconds.

#### 6.50.2.3 EAPI void ecore_x_double_click_time_set (double *t*)

Sets the timeout for a double and triple clicks to be flagged.

This sets the time between clicks before the double_click flag is set in a button down event. If 3 clicks occur within double this time, the triple_click flag is also set.

**Parameters:**

***t*** The time in seconds

#### 6.50.2.4 EAPI int ecore_x_fd_get (void)

Retrieves the X display file descriptor.

**Returns:**

The current X display file descriptor.

## 6.51 X Synchronization Functions

Functions that ensure that all commands that have been issued by the Ecore X library have been sent to the server.

### Functions

- EAPI void ecore_x_flush (void)

  *Sends all X commands in the X Display buffer.*

- EAPI void ecore_x_sync (void)

  *Flushes the command buffer and waits until all requests have been processed by the server.*

### 6.51.1 Detailed Description

Functions that ensure that all commands that have been issued by the Ecore X library have been sent to the server.

## 6.52 X DPMS Extension Functions

Functions related to the X DPMS extension.

### Functions

- EAPI int ecore_x_dpms_query (void)

  *Checks if the X DPMS extension is available on the server.*

- EAPI int ecore_x_dpms_capable_get (void)

  *Checks if the X server is capable of DPMS.*

- EAPI int ecore_x_dpms_enabled_get (void)

  *Checks the DPMS state of the display.*

- EAPI void ecore_x_dpms_enabled_set (int enabled)

  *Sets the DPMS state of the display.*

- EAPI void ecore_x_dpms_timeouts_get (unsigned int *standby, unsigned int *suspend, unsigned int *off)

  *Gets the timeouts.*

- EAPI int ecore_x_dpms_timeouts_set (unsigned int standby, unsigned int suspend, unsigned int off)

  *Sets the timeouts.*

- EAPI unsigned int ecore_x_dpms_timeout_standby_get ()

  *Returns the amount of time of inactivity before standby mode is invoked.*

- EAPI unsigned int ecore_x_dpms_timeout_suspend_get ()

  *Returns the amount of time of inactivity before the second level of power saving is invoked.*

- EAPI unsigned int ecore_x_dpms_timeout_off_get ()

  *Returns the amount of time of inactivity before the third and final level of power saving is invoked.*

- EAPI void ecore_x_dpms_timeout_standby_set (unsigned int new_timeout)

  *Sets the standby timeout (in unit of seconds).*

- EAPI void ecore_x_dpms_timeout_suspend_set (unsigned int new_timeout)

  *Sets the suspend timeout (in unit of seconds).*

- EAPI void ecore_x_dpms_timeout_off_set (unsigned int new_timeout)

  *Sets the off timeout (in unit of seconds).*

### 6.52.1 Detailed Description

Functions related to the X DPMS extension.

### 6.52.2 Function Documentation

#### 6.52.2.1 EAPI int ecore_x_dpms_capable_get (void)

Checks if the X server is capable of DPMS.

**Returns:**

`1` if the X server is capable of DPMS, `0` otherwise.

#### 6.52.2.2 EAPI int ecore_x_dpms_enabled_get (void)

Checks the DPMS state of the display.

**Returns:**

`1` if DPMS is enabled, `0` otherwise.

#### 6.52.2.3 EAPI void ecore_x_dpms_enabled_set (int *enabled*)

Sets the DPMS state of the display.

**Parameters:**

***enabled*** `0` to disable DPMS characteristics of the server, enable it otherwise.

#### 6.52.2.4 EAPI int ecore_x_dpms_query (void)

Checks if the X DPMS extension is available on the server.

**Returns:**

`1` if the X DPMS extension is available, `0` otherwise.

#### 6.52.2.5 EAPI unsigned int ecore_x_dpms_timeout_off_get ()

Returns the amount of time of inactivity before the third and final level of power saving is invoked.

**Returns:**

The off timeout value.

#### 6.52.2.6 EAPI void ecore_x_dpms_timeout_off_set (unsigned int *new_timeout*)

Sets the off timeout (in unit of seconds).

**Parameters:**

***off*** Amount of time of inactivity before the monitor is shut off.

#### 6.52.2.7 EAPI unsigned int ecore_x_dpms_timeout_standby_get ()

Returns the amount of time of inactivity before standby mode is invoked.

**Returns:**

The standby timeout value.

#### 6.52.2.8 EAPI void ecore_x_dpms_timeout_standby_set (unsigned int *new_timeout*)

Sets the standby timeout (in unit of seconds).

**Parameters:**

***new_standby*** Amount of time of inactivity before standby mode will be invoked.

#### 6.52.2.9 EAPI unsigned int ecore_x_dpms_timeout_suspend_get ()

Returns the amount of time of inactivity before the second level of power saving is invoked.

**Returns:**

The suspend timeout value.

#### 6.52.2.10 EAPI void ecore_x_dpms_timeout_suspend_set (unsigned int *new_timeout*)

Sets the suspend timeout (in unit of seconds).

**Parameters:**

***suspend*** Amount of time of inactivity before the screen is placed into suspend mode.

#### 6.52.2.11 EAPI void ecore_x_dpms_timeouts_get (unsigned int * *standby*, unsigned int * *suspend*, unsigned int * *off*)

Gets the timeouts.

The values are in unit of seconds.

**Parameters:**

***standby*** Amount of time of inactivity before standby mode will be invoked.

***suspend*** Amount of time of inactivity before the screen is placed into suspend mode.

***off*** Amount of time of inactivity before the monitor is shut off.

#### 6.52.2.12 EAPI int ecore_x_dpms_timeouts_set (unsigned int *standby*, unsigned int *suspend*, unsigned int *off*)

Sets the timeouts.

The values are in unit of seconds.

**Parameters:**

***standby*** Amount of time of inactivity before standby mode will be invoked.

***suspend*** Amount of time of inactivity before the screen is placed into suspend mode.

***off*** Amount of time of inactivity before the monitor is shut off.

## 6.53 X Drawable Functions

Functions that operate on drawables.

### Functions

- EAPI void ecore_x_drawable_geometry_get (Ecore_X_Drawable d, int *x, int *y, int *w, int *h)

  *Retrieves the geometry of the given drawable.*

- EAPI int ecore_x_drawable_border_width_get (Ecore_X_Drawable d)

  *Retrieves the width of the border of the given drawable.*

- EAPI int ecore_x_drawable_depth_get (Ecore_X_Drawable d)

  *Retrieves the depth of the given drawable.*

### 6.53.1 Detailed Description

Functions that operate on drawables.

### 6.53.2 Function Documentation

#### 6.53.2.1 EAPI int ecore_x_drawable_border_width_get (Ecore_X_Drawable *d*)

Retrieves the width of the border of the given drawable.

**Parameters:**

***d*** The given drawable.

**Returns:**

The border width of the given drawable.

#### 6.53.2.2 EAPI int ecore_x_drawable_depth_get (Ecore_X_Drawable *d*)

Retrieves the depth of the given drawable.

**Parameters:**

***d*** The given drawable.

**Returns:**

The depth of the given drawable.

#### 6.53.2.3 EAPI void ecore_x_drawable_geometry_get (Ecore_X_Drawable *d*, int * *x*, int * *y*, int * *w*, int * *h*)

Retrieves the geometry of the given drawable.

**Parameters:**

***d*** The given drawable.

***x*** Pointer to an integer into which the X position is to be stored.

***y*** Pointer to an integer into which the Y position is to be stored.

***w*** Pointer to an integer into which the width is to be stored.

***h*** Pointer to an integer into which the height is to be stored.

## 6.54 X Pixmap Functions

Functions that operate on pixmaps.

### Functions

- EAPI Ecore_X_Pixmap ecore_x_pixmap_new (Ecore_X_Window win, int w, int h, int dep)

  *Creates a new pixmap.*

- EAPI void ecore_x_pixmap_del (Ecore_X_Pixmap pmap)

  *Deletes the reference to the given pixmap.*

- EAPI void ecore_x_pixmap_paste (Ecore_X_Pixmap pmap, Ecore_X_Drawable dest, Ecore_X_GC gc, int sx, int sy, int w, int h, int dx, int dy)

  *Pastes a rectangular area of the given pixmap onto the given drawable.*

- EAPI void ecore_x_pixmap_geometry_get (Ecore_X_Pixmap pmap, int *x, int *y, int *w, int *h)

  *Retrieves the size of the given pixmap.*

- EAPI int ecore_x_pixmap_depth_get (Ecore_X_Pixmap pmap)

  *Retrieves the depth of the given pixmap.*

### 6.54.1 Detailed Description

Functions that operate on pixmaps.

### 6.54.2 Function Documentation

#### 6.54.2.1 EAPI void ecore_x_pixmap_del (Ecore_X_Pixmap *pmap*)

Deletes the reference to the given pixmap.

If no other clients have a reference to the given pixmap, the server will destroy it.

**Parameters:**

***pmap*** The given pixmap.

#### 6.54.2.2 EAPI int ecore_x_pixmap_depth_get (Ecore_X_Pixmap *pmap*)

Retrieves the depth of the given pixmap.

**Parameters:**

***pmap*** The given pixmap.

**Returns:**

The depth of the pixmap.

#### 6.54.2.3 EAPI void ecore_x_pixmap_geometry_get (Ecore_X_Pixmap *pmap*, int * *x*, int * *y*, int * *w*, int * *h*)

Retrieves the size of the given pixmap.

**Parameters:**

***pmap*** The given pixmap.

***x*** Pointer to an integer in which to store the X position.

***y*** Pointer to an integer in which to store the Y position.

***w*** Pointer to an integer in which to store the width.

***h*** Pointer to an integer in which to store the height.

#### 6.54.2.4 EAPI Ecore_X_Pixmap ecore_x_pixmap_new (Ecore_X_Window *win*, int *w*, int *h*, int *dep*)

Creates a new pixmap.

**Parameters:**

***win*** Window used to determine which screen of the display the pixmap should be created on. If 0, the default root window is used.

***w*** Width of the new pixmap.

***h*** Height of the new pixmap.

***dep*** Depth of the pixmap. If 0, the default depth of the default screen is used.

**Returns:**

New pixmap.

#### 6.54.2.5 EAPI void ecore_x_pixmap_paste (Ecore_X_Pixmap *pmap*, Ecore_X_Drawable *dest*, Ecore_X_GC *gc*, int *sx*, int *sy*, int *w*, int *h*, int *dx*, int *dy*)

Pastes a rectangular area of the given pixmap onto the given drawable.

**Parameters:**

***pmap*** The given pixmap.

***dest*** The given drawable.

***gc*** The graphics context which governs which operation will be used to paste the area onto the drawable.

***sx*** The X position of the area on the pixmap.

***sy*** The Y position of the area on the pixmap.

***w*** The width of the area.

***h*** The height of the area.

***dx*** The X position at which to paste the area on `dest`.

***dy*** The Y position at which to paste the area on `dest`.

## 6.55 X Window Creation Functions

Functions that can be used to create an X window.

### Functions

- EAPI Ecore_X_Window ecore_x_window_new (Ecore_X_Window parent, int x, int y, int w, int h)

  *Creates a new window.*

- EAPI Ecore_X_Window ecore_x_window_override_new (Ecore_X_Window parent, int x, int y, int w, int h)

  *Creates a window with the override redirect attribute set to* `True`.

- EAPI Ecore_X_Window ecore_x_window_input_new (Ecore_X_Window parent, int x, int y, int w, int h)

  *Creates a new input window.*

- EAPI Ecore_X_Window ecore_x_window_manager_argb_new (Ecore_X_Window parent, int x, int y, int w, int h)

  *Creates a new window.*

- EAPI Ecore_X_Window ecore_x_window_argb_new (Ecore_X_Window parent, int x, int y, int w, int h)

  *Creates a new window.*

- EAPI Ecore_X_Window ecore_x_window_override_argb_new (Ecore_X_Window parent, int x, int y, int w, int h)

  *Creates a window with the override redirect attribute set to* `True`.

### 6.55.1 Detailed Description

Functions that can be used to create an X window.

### 6.55.2 Function Documentation

#### 6.55.2.1 EAPI Ecore_X_Window ecore_x_window_argb_new (Ecore_X_Window *parent*, int *x*, int *y*, int *w*, int *h*)

Creates a new window.

**Parameters:**

***parent*** The parent window to use. If `parent` is `0`, the root window of the default display is used.

***x*** X position.

***y*** Y position.

***w*** Width.

***h*** Height.

**Returns:**

The new window handle.

#### 6.55.2.2 EAPI Ecore_X_Window ecore_x_window_input_new (Ecore_X_Window *parent*, int *x*, int *y*, int *w*, int *h*)

Creates a new input window.

**Parameters:**

***parent*** The parent window to use. If `parent` is `0`, the root window of the default display is used.

***x*** X position.

***y*** Y position.

***w*** Width.

***h*** Height.

**Returns:**

The new window.

#### 6.55.2.3 EAPI Ecore_X_Window ecore_x_window_manager_argb_new (Ecore_X_Window *parent*, int *x*, int *y*, int *w*, int *h*)

Creates a new window.

**Parameters:**

***parent*** The parent window to use. If `parent` is `0`, the root window of the default display is used.

***x*** X position.

***y*** Y position.

***w*** Width.

***h*** Height.

**Returns:**

The new window handle.

#### 6.55.2.4 EAPI Ecore_X_Window ecore_x_window_new (Ecore_X_Window *parent*, int *x*, int *y*, int *w*, int *h*)

Creates a new window.

**Parameters:**

***parent*** The parent window to use. If `parent` is `0`, the root window of the default display is used.

***x*** X position.

***y*** Y position.

***w*** Width.

***h*** Height.

**Returns:**

The new window handle.

#### 6.55.2.5 EAPI Ecore_X_Window ecore_x_window_override_argb_new (Ecore_X_Window *parent*, int *x*, int *y*, int *w*, int *h*)

Creates a window with the override redirect attribute set to `True`.

**Parameters:**

***parent*** The parent window to use. If `parent` is `0`, the root window of the default display is used.

***x*** X position.

***y*** Y position.

***w*** Width.

***h*** Height.

**Returns:**

The new window handle.

#### 6.55.2.6 EAPI Ecore_X_Window ecore_x_window_override_new (Ecore_X_Window *parent*, int *x*, int *y*, int *w*, int *h*)

Creates a window with the override redirect attribute set to `True`.

**Parameters:**

***parent*** The parent window to use. If `parent` is `0`, the root window of the default display is used.

***x*** X position.

***y*** Y position.

***w*** Width.

***h*** Height.

**Returns:**

The new window handle.

## 6.56 X Window Property Functions

Functions that set window properties.

## 6.57 X Window Destroy Functions

Functions to destroy X windows.

### Functions

- EAPI void ecore_x_window_del (Ecore_X_Window win)
  *Deletes the given window.*
- EAPI void ecore_x_window_delete_request_send (Ecore_X_Window win)
  *Sends a delete request to the given window.*

### 6.57.1 Detailed Description

Functions to destroy X windows.

### 6.57.2 Function Documentation

#### 6.57.2.1 EAPI void ecore_x_window_del (Ecore_X_Window *win*)

Deletes the given window.

**Parameters:**

***win*** The given window.

#### 6.57.2.2 EAPI void ecore_x_window_delete_request_send (Ecore_X_Window *win*)

Sends a delete request to the given window.

**Parameters:**

***win*** The given window.

## 6.58 X Window Visibility Functions

Functions to access and change the visibility of X windows.

## 6.59 X Window Geometry Functions

Functions that change or retrieve the geometry of X windows.

### Functions

- EAPI void ecore_x_window_move (Ecore_X_Window win, int x, int y)

  *Moves a window to the position* `x`, `y`.

- EAPI void ecore_x_window_resize (Ecore_X_Window win, int w, int h)

  *Resizes a window.*

- EAPI void ecore_x_window_move_resize (Ecore_X_Window win, int x, int y, int w, int h)

  *Moves and resizes a window.*

- EAPI void ecore_x_window_size_get (Ecore_X_Window win, int *w, int *h)

  *Retrieves the size of the given window.*

- EAPI void ecore_x_window_geometry_get (Ecore_X_Window win, int *x, int *y, int *w, int *h)

  *Retrieves the geometry of the given window.*

- EAPI int ecore_x_window_border_width_get (Ecore_X_Window win)

  *Retrieves the width of the border of the given window.*

- EAPI void ecore_x_window_border_width_set (Ecore_X_Window win, int width)

  *Sets the width of the border of the given window.*

- EAPI Ecore_X_Window ecore_x_window_at_xy_get (int x, int y)

  *Retrieves the top, visible window at the given location.*

- EAPI Ecore_X_Window ecore_x_window_at_xy_with_skip_get (int x, int y, Ecore_X_Window *skip, int skip_num)

  *Retrieves the top, visible window at the given location, but skips the windows in the list.*

### 6.59.1 Detailed Description

Functions that change or retrieve the geometry of X windows.

### 6.59.2 Function Documentation

#### 6.59.2.1 EAPI Ecore_X_Window ecore_x_window_at_xy_get (int *x*, int *y*)

Retrieves the top, visible window at the given location.

**Parameters:**

***x*** The given X position.
***y*** The given Y position.

**Returns:**

The window at that position.

#### 6.59.2.2 EAPI Ecore_X_Window ecore_x_window_at_xy_with_skip_get (int *x*, int *y*, Ecore_X_Window * *skip*, int *skip_num*)

Retrieves the top, visible window at the given location, but skips the windows in the list.

**Parameters:**

***x*** The given X position.
***y*** The given Y position.

**Returns:**

The window at that position.

#### 6.59.2.3 EAPI int ecore_x_window_border_width_get (Ecore_X_Window *win*)

Retrieves the width of the border of the given window.

**Parameters:**

***win*** The given window.

**Returns:**

Width of the border of `win`.

#### 6.59.2.4 EAPI void ecore_x_window_border_width_set (Ecore_X_Window *win*, int *width*)

Sets the width of the border of the given window.

**Parameters:**

***win*** The given window.
***width*** The new border width.

#### 6.59.2.5 EAPI void ecore_x_window_geometry_get (Ecore_X_Window *win*, int * *x*, int * *y*, int * *w*, int * *h*)

Retrieves the geometry of the given window.

**Parameters:**

***win*** The given window.

***x*** Pointer to an integer in which the X position is to be stored.

***y*** Pointer to an integer in which the Y position is to be stored.

***w*** Pointer to an integer in which the width is to be stored.

***h*** Pointer to an integer in which the height is to be stored.

#### 6.59.2.6 EAPI void ecore_x_window_move (Ecore_X_Window *win*, int *x*, int *y*)

Moves a window to the position `x`, `y`.

The position is relative to the upper left hand corner of the parent window.

**Parameters:**

***win*** The window to move.

***x*** X position.

***y*** Y position.

#### 6.59.2.7 EAPI void ecore_x_window_move_resize (Ecore_X_Window *win*, int *x*, int *y*, int *w*, int *h*)

Moves and resizes a window.

**Parameters:**

***win*** The window to move and resize.

***x*** New X position of the window.

***y*** New Y position of the window.

***w*** New width of the window.

***h*** New height of the window.

#### 6.59.2.8 EAPI void ecore_x_window_resize (Ecore_X_Window *win*, int *w*, int *h*)

Resizes a window.

**Parameters:**

***win*** The window to resize.

***w*** New width of the window.

***h*** New height of the window.

#### 6.59.2.9 EAPI void ecore_x_window_size_get (Ecore_X_Window *win*, int * *w*, int * *h*)

Retrieves the size of the given window.

**Parameters:**

***win*** The given window.

***w*** Pointer to an integer into which the width is to be stored.

***h*** Pointer to an integer into which the height is to be stored.

## 6.60 X Window Focus Functions

Functions that give the focus to an X Window.

### Functions

- EAPI void ecore_x_window_focus (Ecore_X_Window win)

  *Sets the focus to the window* `win`.

- EAPI void ecore_x_window_focus_at_time (Ecore_X_Window win, Ecore_X_Time t)

  *Sets the focus to the given window at a specific time.*

- EAPI Ecore_X_Window ecore_x_window_focus_get (void)

  *gets the focus to the window* `win`.

### 6.60.1 Detailed Description

Functions that give the focus to an X Window.

### 6.60.2 Function Documentation

#### 6.60.2.1 EAPI void ecore_x_window_focus (Ecore_X_Window *win*)

Sets the focus to the window `win`.

**Parameters:**

***win*** The window to focus.

#### 6.60.2.2 EAPI void ecore_x_window_focus_at_time (Ecore_X_Window *win*, Ecore_X_Time *t*)

Sets the focus to the given window at a specific time.

**Parameters:**

***win*** The window to focus.

***t*** When to set the focus to the window.

#### 6.60.2.3 EAPI Ecore_X_Window ecore_x_window_focus_get (void)

gets the focus to the window `win`.

**Returns:**

The window that has focus.

## 6.61 X Window Z Order Functions

Functions that change the Z order of X windows.

### Functions

- EAPI void ecore_x_window_raise (Ecore_X_Window win)
  *Raises the given window.*
- EAPI void ecore_x_window_lower (Ecore_X_Window win)
  *Lowers the given window.*

### 6.61.1 Detailed Description

Functions that change the Z order of X windows.

### 6.61.2 Function Documentation

#### 6.61.2.1 EAPI void ecore_x_window_lower (Ecore_X_Window *win*)

Lowers the given window.

**Parameters:**

***win*** The window to lower.

#### 6.61.2.2 EAPI void ecore_x_window_raise (Ecore_X_Window *win*)

Raises the given window.

**Parameters:**

***win*** The window to raise.

## 6.62 X Window Parent Functions

Functions that retrieve or changes the parent window of a window.

### Functions

- EAPI void ecore_x_window_reparent (Ecore_X_Window win, Ecore_X_Window new_parent, int x, int y)

  *Moves a window to within another window at a given position.*

- EAPI Ecore_X_Window ecore_x_window_parent_get (Ecore_X_Window win)

  *Retrieves the parent window of the given window.*

### 6.62.1 Detailed Description

Functions that retrieve or changes the parent window of a window.

### 6.62.2 Function Documentation

#### 6.62.2.1 EAPI Ecore_X_Window ecore_x_window_parent_get (Ecore_X_Window *win*)

Retrieves the parent window of the given window.

**Parameters:**

***win*** The given window.

**Returns:**

The parent window of `win`.

#### 6.62.2.2 EAPI void ecore_x_window_reparent (Ecore_X_Window *win*, Ecore_X_Window *new_parent*, int *x*, int *y*)

Moves a window to within another window at a given position.

**Parameters:**

***win*** The window to reparent.
***new_parent*** The new parent window.
***x*** X position within new parent window.
***y*** Y position within new parent window.

## 6.63 X Window Shape Functions

These functions use the shape extension of the X server to change shape of given windows.

### Functions

- EAPI void ecore_x_window_shape_mask_set (Ecore_X_Window win, Ecore_X_Pixmap mask)

  *Sets the shape of the given window to that given by the pixmap* `mask`.

### 6.63.1 Detailed Description

These functions use the shape extension of the X server to change shape of given windows.

### 6.63.2 Function Documentation

#### 6.63.2.1 EAPI void ecore_x_window_shape_mask_set (Ecore_X_Window *win*, Ecore_X_Pixmap *mask*)

Sets the shape of the given window to that given by the pixmap `mask`.

**Parameters:**

***win*** The given window.

***mask*** A 2-bit depth pixmap that provides the new shape of the window.

# Chapter 7

# Ecore Data Structure Documentation

## 7.1 _Ecore_DirectFB_Event_Key_Down Struct Reference

DirectFB Key Down event.

### Data Fields

- char * name
  *The name of the key that was released.*
- char * string
  *The logical symbol of the key that was pressed.*
- char * key_compose
  *The UTF-8 string conversion if any.*

### 7.1.1 Detailed Description

DirectFB Key Down event.

## 7.2 _Ecore_DirectFB_Event_Key_Up Struct Reference

DirectFB Key Up event.

### Data Fields

- char * name
  *The name of the key that was released.*
- char * string
  *The logical symbol of the key that was pressed.*
- char * key_compose
  *The UTF-8 string conversion if any.*

### 7.2.1 Detailed Description

DirectFB Key Up event.

## 7.3 _Ecore_Event_Signal_Exit Struct Reference

Exit request event.

### Data Fields

- int interrupt: 1
  *Set if the exit request was an interrupt signal.*
- int quit: 1
  *set if the exit request was a quit signal*
- int terminate: 1
  *Set if the exit request was a terminate singal.*
- void * ext_data
  *Extension data - not used.*
- siginfo_t data
  *Signal info.*

### 7.3.1 Detailed Description

Exit request event.

## 7.4 _Ecore_Event_Signal_Hup Struct Reference

Hup signal event.

### Data Fields

- void * ext_data
  *Extension data - not used.*
- siginfo_t data
  *Signal info.*

### 7.4.1 Detailed Description

Hup signal event.

## 7.5 _Ecore_Event_Signal_Power Struct Reference

Power event.

### Data Fields

- void * ext_data
  *Extension data - not used.*
- siginfo_t data
  *Signal info.*

### 7.5.1 Detailed Description

Power event.

## 7.6 _Ecore_Event_Signal_Realtime Struct Reference

Realtime event.

### Data Fields

- int num
  *The realtime signal's number.*
- siginfo_t data
  *Signal info.*

### 7.6.1 Detailed Description

Realtime event.

# 7.7 _Ecore_Event_Signal_User Struct Reference

User signal event.

## Data Fields

- int number
  *The signal number.*
- void * ext_data
  *Extension data - not used.*
- siginfo_t data
  *Signal info.*

### 7.7.1 Detailed Description

User signal event.

### 7.7.2 Field Documentation

#### 7.7.2.1 int _Ecore_Event_Signal_User::number

The signal number.

Either 1 or 2

## 7.8 _Ecore_Exe_Event_Add Struct Reference

Process add event.

### Data Fields

- Ecore_Exe * exe
  *The handle to the added process.*
- void * ext_data
  *Extension data - not used.*

### 7.8.1 Detailed Description

Process add event.

## 7.9 _Ecore_Exe_Event_Data Struct Reference

Data from a child process event.

### Data Fields

- Ecore_Exe * exe
  *The handle to the process.*
- void * data
  *the raw binary data from the child process that was recieved*
- int size
  *the size of this data in bytes*
- Ecore_Exe_Event_Data_Line * lines
  *an array of line data if line buffered, the last one has it's line member set to NULL*

### 7.9.1 Detailed Description

Data from a child process event.

## 7.10 _Ecore_Exe_Event_Data_Line Struct Reference

< Lines from a child process

### 7.10.1 Detailed Description

< Lines from a child process

## 7.11 _Ecore_Exe_Event_Del Struct Reference

Process exit event.

### Data Fields

- pid_t pid
  *The process ID of the process that exited.*
- int exit_code
  *The exit code of the process.*
- Ecore_Exe * exe
  *The handle to the exited process, or NULL if not found.*
- char exited: 1
  *< The signal that caused the process to exit*
- char signalled: 1
  *< set to 1 if the process exited of its own accord*
- void * ext_data
  *< set to 1 id the process exited due to uncaught signal Extension data - not used*
- siginfo_t data
  *Signal info.*

### 7.11.1 Detailed Description

Process exit event.

## 7.12 _Ecore_Fb_Event_Key_Down Struct Reference

FB Key Down event.

### Data Fields

- Ecore_Fb_Input_Device * dev
  *The device associated with the event.*
- char * keyname
  *The name of the key that was pressed.*
- char * keysymbol
  *The logical symbol of the key that was pressed.*
- char * key_compose
  *The UTF-8 string conversion if any.*

### 7.12.1 Detailed Description

FB Key Down event.

## 7.13 _Ecore_Fb_Event_Key_Up Struct Reference

FB Key Up event.

### Data Fields

- Ecore_Fb_Input_Device * dev
  *The device associated with the event.*
- char * keyname
  *The name of the key that was released.*
- char * keysymbol
  *The logical symbol of the key that was pressed.*
- char * key_compose
  *The UTF-8 string conversion if any.*

### 7.13.1 Detailed Description

FB Key Up event.

## 7.14 _Ecore_Fb_Event_Mouse_Button_Down Struct Reference

FB Mouse Down event.

### Data Fields

- Ecore_Fb_Input_Device * dev
  *The device associated with the event.*
- int button
  *Mouse button that was pressed (1 - 32).*
- int x
  *Mouse co-ordinates when mouse button was pressed.*
- int y
  *Mouse co-ordinates when mouse button was pressed.*
- int double_click: 1
  *Set if click was a double click.*
- int triple_click: 1
  *Set if click was a triple click.*

### 7.14.1 Detailed Description

FB Mouse Down event.

## 7.15 _Ecore_Fb_Event_Mouse_Button_Up Struct Reference

FB Mouse Up event.

### Data Fields

- Ecore_Fb_Input_Device * dev
  *The device associated with the event.*
- int button
  *Mouse button that was released (1 - 32).*
- int x
  *Mouse co-ordinates when mouse button was raised.*
- int y
  *Mouse co-ordinates when mouse button was raised.*

### 7.15.1 Detailed Description

FB Mouse Up event.

## 7.16 _Ecore_Fb_Event_Mouse_Move Struct Reference

FB Mouse Move event.

### Data Fields

- Ecore_Fb_Input_Device * dev
  *The device associated with the event.*
- int x
  *Mouse co-ordinates where the mouse cursor moved to.*
- int y
  *Mouse co-ordinates where the mouse cursor moved to.*

### 7.16.1 Detailed Description

FB Mouse Move event.

# 7.17 _Ecore_Fb_Event_Mouse_Wheel Struct Reference

FB Mouse Wheel event.

## 7.17.1 Detailed Description

FB Mouse Wheel event.

## 7.18 Ecore_Config_Prop Struct Reference

The actual property for storing a key-value pair.

### Data Fields

- Ecore_Config_Listener_List * listeners
  < *Configuration flags.*
- Ecore_Config_Prop * parent
  < *Stores extra data for the property.*

### 7.18.1 Detailed Description

The actual property for storing a key-value pair.

# Chapter 8

# Ecore File Documentation

## 8.1 Ecore.h File Reference

The file that provides the program utility, main loop and timer functions.

### Data Structures

- struct _Ecore_Event_Signal_User
  *User signal event.*
- struct _Ecore_Event_Signal_Hup
  *Hup signal event.*
- struct _Ecore_Event_Signal_Exit
  *Exit request event.*
- struct _Ecore_Event_Signal_Power
  *Power event.*
- struct _Ecore_Event_Signal_Realtime
  *Realtime event.*
- struct _Ecore_Exe_Event_Add
  *Process add event.*
- struct _Ecore_Exe_Event_Del
  *Process exit event.*
- struct _Ecore_Exe_Event_Data_Line
  *< Lines from a child process*
- struct _Ecore_Exe_Event_Data
  *Data from a child process event.*

## Defines

- #define ECORE_EVENT_SIGNAL_USER 1
  *User signal event.*
- #define ECORE_EVENT_SIGNAL_HUP 2
  *Hup signal event.*
- #define ECORE_EVENT_SIGNAL_EXIT 3
  *Exit signal event.*
- #define ECORE_EVENT_SIGNAL_POWER 4
  *Power signal event.*
- #define ECORE_EVENT_SIGNAL_REALTIME 5
  *Realtime signal event.*

## Typedefs

- typedef void Ecore_Exe
  *A handle for spawned processes.*
- typedef void Ecore_Timer
  *A handle for timers.*
- typedef void Ecore_Idler
  *A handle for idlers.*
- typedef void Ecore_Idle_Enterer
  *A handle for idle enterers.*
- typedef void Ecore_Idle_Exiter
  *A handle for idle exiters.*
- typedef void Ecore_Fd_Handler
  *A handle for Fd hanlders.*
- typedef void Ecore_Event_Handler
  *A handle for an event handler.*
- typedef void Ecore_Event_Filter
  *A handle for an event filter.*
- typedef void Ecore_Event
  *A handle for an event.*
- typedef void Ecore_Animator
  *A handle for animators.*

- typedef _Ecore_Event_Signal_User Ecore_Event_Signal_User
  *User signal event.*
- typedef _Ecore_Event_Signal_Hup Ecore_Event_Signal_Hup
  *Hup signal event.*
- typedef _Ecore_Event_Signal_Exit Ecore_Event_Signal_Exit
  *Exit signal event.*
- typedef _Ecore_Event_Signal_Power Ecore_Event_Signal_Power
  *Power signal event.*
- typedef _Ecore_Event_Signal_Realtime Ecore_Event_Signal_Realtime
  *Realtime signal event.*
- typedef _Ecore_Exe_Event_Add Ecore_Exe_Event_Add
  *Spawned Exe add event.*
- typedef _Ecore_Exe_Event_Del Ecore_Exe_Event_Del
  *Spawned Exe exit event.*
- typedef _Ecore_Exe_Event_Data_Line Ecore_Exe_Event_Data_Line
  *Lines from a child process.*
- typedef _Ecore_Exe_Event_Data Ecore_Exe_Event_Data
  *Data from a child process.*

## Enumerations

- enum _Ecore_Fd_Handler_Flags {
  ECORE_FD_READ = 1,
  ECORE_FD_WRITE = 2,
  ECORE_FD_ERROR = 4 }
- enum _Ecore_Exe_Flags {
  ECORE_EXE_PIPE_READ = 1,
  ECORE_EXE_PIPE_WRITE = 2,
  ECORE_EXE_PIPE_ERROR = 4,
  ECORE_EXE_PIPE_READ_LINE_BUFFERED = 8,
  ECORE_EXE_PIPE_ERROR_LINE_BUFFERED = 16,
  ECORE_EXE_PIPE_AUTO = 32,
  ECORE_EXE_RESPAWN = 64,
  ECORE_EXE_USE_SH = 128 }

## Functions

- EAPI int ecore_init (void)

  *Set up connections, signal handlers, sockets etc.*

- EAPI int ecore_shutdown (void)

  *Shut down connections, signal handlers sockets etc.*

- EAPI void ecore_app_args_set (int argc, const char **argv)

  *Set up the programs command-line arguments.*

- EAPI void ecore_app_args_get (int *argc, char ***argv)

  *Return the programs stored command-line arguments.*

- EAPI void ecore_app_restart (void)

  *Restart the program executable with the command-line arguments stored.*

- EAPI Ecore_Event_Handler * ecore_event_handler_add (int type, int(*func)(void *data, int type, void *event), const void *data)

  *Add an event handler.*

- EAPI void * ecore_event_handler_del (Ecore_Event_Handler *event_handler)

  *Delete an event handler.*

- EAPI Ecore_Event * ecore_event_add (int type, void *ev, void(*func_free)(void *data, void *ev), void *data)

  *Add an event to the event queue.*

- EAPI void * ecore_event_del (Ecore_Event *event)

  *Delete an event from the queue.*

- EAPI int ecore_event_type_new (void)

  *Allocate a new event type id sensibly and return the new id.*

- EAPI Ecore_Event_Filter * ecore_event_filter_add (void *(*func_start)(void *data), int(*func_filter)(void *data, void *loop_data, int type, void *event), void(*func_end)(void *data, void *loop_data), const void *data)

  *Add a filter the current event queue.*

- EAPI void * ecore_event_filter_del (Ecore_Event_Filter *ef)

  *Delete an event filter.*

- EAPI int ecore_event_current_type_get (void)

  *Return the current event type being handled.*

- EAPI void * ecore_event_current_event_get (void)

  *Return the current event type pointer handled.*

- EAPI Ecore_Exe * ecore_exe_run (const char *exe_cmd, const void *data)

  *Spawns a child process.*

- EAPI Ecore_Exe * ecore_exe_pipe_run (const char *exe_cmd, Ecore_Exe_Flags flags, const void *data)

  *Spawns a child process with its stdin/out available for communication.*

- EAPI int ecore_exe_send (Ecore_Exe *exe, void *data, int size)

  *Sends data to the given child process which it recieves on stdin.*

- EAPI void ecore_exe_close_stdin (Ecore_Exe *exe)

  *The stdin of the given child process will close when the write buffer is empty.*

- EAPI void ecore_exe_auto_limits_set (Ecore_Exe *exe, int start_bytes, int end_bytes, int start_lines, int end_lines)

  *Sets the auto pipe limits for the given process handle.*

- EAPI Ecore_Exe_Event_Data * ecore_exe_event_data_get (Ecore_Exe *exe, Ecore_Exe_Flags flags)

  *Gets the auto pipe data for the given process handle.*

- EAPI void ecore_exe_event_data_free (Ecore_Exe_Event_Data *data)

  *Frees the given event data.*

- EAPI void * ecore_exe_free (Ecore_Exe *exe)

  *Frees the given process handle.*

- EAPI pid_t ecore_exe_pid_get (Ecore_Exe *exe)

  *Retrieves the process ID of the given spawned process.*

- EAPI void ecore_exe_tag_set (Ecore_Exe *exe, const char *tag)

  *Sets the string tag for the given process handle.*

- EAPI char * ecore_exe_tag_get (Ecore_Exe *exe)

  *Retrieves the tag attached to the given process handle.*

- EAPI char * ecore_exe_cmd_get (Ecore_Exe *exe)

  *Retrieves the command of the given spawned process.*

- EAPI void * ecore_exe_data_get (Ecore_Exe *exe)

  *Retrieves the data attached to the given process handle.*

- EAPI void ecore_exe_pause (Ecore_Exe *exe)

  *Pauses the given process by sending it a* `SIGSTOP` *signal.*

- EAPI void ecore_exe_continue (Ecore_Exe *exe)

  *Continues the given paused process by sending it a* `SIGCONT` *signal.*

- EAPI void ecore_exe_terminate (Ecore_Exe *exe)

  *Sends the given spawned process a terminate (*`SIGTERM`*) signal.*

- EAPI void ecore_exe_kill (Ecore_Exe *exe)

*Kills the given spawned process by sending it a* `SIGKILL` *signal.*

- EAPI void ecore_exe_signal (Ecore_Exe *exe, int num)

  *Sends a* `SIGUSR` *signal to the given spawned process.*

- EAPI void ecore_exe_hup (Ecore_Exe *exe)

  *Sends a* `SIGHUP` *signal to the given spawned process.*

- EAPI Ecore_Idler * ecore_idler_add (int(*func)(void *data), const void *data)

  *Add an idler handler.*

- EAPI void * ecore_idler_del (Ecore_Idler *idler)

  *Delete an idler callback from the list to be executed.*

- EAPI Ecore_Idle_Enterer * ecore_idle_enterer_add (int(*func)(void *data), const void *data)

  *Add an idle enterer handler.*

- EAPI void * ecore_idle_enterer_del (Ecore_Idle_Enterer *idle_enterer)

  *Delete an idle enterer callback.*

- EAPI Ecore_Idle_Exiter * ecore_idle_exiter_add (int(*func)(void *data), const void *data)

  *Add an idle exiter handler.*

- EAPI void * ecore_idle_exiter_del (Ecore_Idle_Exiter *idle_exiter)

  *Delete an idle exiter handler from the list to be run on exiting idle state.*

- EAPI void ecore_main_loop_iterate (void)

  *Runs a single iteration of the main loop to process everything on the queue.*

- EAPI void ecore_main_loop_begin (void)

  *Runs the application main loop.*

- EAPI void ecore_main_loop_quit (void)

  *Quits the main loop once all the events currently on the queue have been processed.*

- EAPI Ecore_Fd_Handler * ecore_main_fd_handler_add (int fd, Ecore_Fd_Handler_Flags flags, int(*func)(void *data, Ecore_Fd_Handler *fd_handler), const void *data, int(*buf_func)(void *buf_data, Ecore_Fd_Handler *fd_handler), const void *buf_data)

  *Adds a callback for activity on the given file descriptor.*

- EAPI void * ecore_main_fd_handler_del (Ecore_Fd_Handler *fd_handler)

  *Deletes the given FD handler.*

- EAPI int ecore_main_fd_handler_fd_get (Ecore_Fd_Handler *fd_handler)

  *Retrieves the file descriptor that the given handler is handling.*

- EAPI int ecore_main_fd_handler_active_get (Ecore_Fd_Handler *fd_handler, Ecore_Fd_Handler_Flags flags)

*Return if read, write or error, or a combination thereof, is active on the file descriptor of the given FD handler.*

- EAPI void ecore_main_fd_handler_active_set (Ecore_Fd_Handler *fd_handler, Ecore_Fd_Handler_Flags flags)

  *Set what active streams the given FD handler should be monitoring.*

- EAPI double ecore_time_get (void)

  *Retrieves the current system time as a floating point value in seconds.*

- EAPI Ecore_Timer * ecore_timer_add (double in, int(*func)(void *data), const void *data)

  *Creates a timer to call the given function in the given period of time.*

- EAPI void * ecore_timer_del (Ecore_Timer *timer)

  *Delete the specified timer from the timer list.*

- EAPI void ecore_timer_interval_set (Ecore_Timer *timer, double in)

  *Change the interval the timer ticks of.*

- EAPI Ecore_Animator * ecore_animator_add (int(*func)(void *data), const void *data)

  *Add a animator to tick off at every animaton tick during main loop execution.*

- EAPI void * ecore_animator_del (Ecore_Animator *animator)

  *Delete the specified animator from the animator list.*

- EAPI void ecore_animator_frametime_set (double frametime)

  *Set the animator call interval in seconds.*

- EAPI double ecore_animator_frametime_get (void)

  *Get the animator call interval in seconds.*

## Variables

- EAPI int ECORE_EXE_EVENT_ADD

  *A child process has been added.*

- EAPI int ECORE_EXE_EVENT_DEL

  *A child process has been deleted (it exited, naming consistant with the rest of ecore).*

- EAPI int ECORE_EXE_EVENT_DATA

  *Data from a child process.*

- EAPI int ECORE_EXE_EVENT_ERROR

  *Errors from a child process.*

### 8.1.1 Detailed Description

The file that provides the program utility, main loop and timer functions.

This header provides the Ecore event handling loop. For more details, see Main Loop Functions.

For the main loop to be of any use, you need to be able to add events and event handlers. Events for file descriptor events are covered in File Event Handling Functions.

Time functions are covered in Ecore Time Functions.

There is also provision for callbacks for when the loop enters or exits an idle state. See Idle Handlers for more information.

Functions are also provided for spawning child processes using fork. See Process Spawning Functions and Spawned Process Signal Functions for more details.

### 8.1.2 Enumeration Type Documentation

#### 8.1.2.1 enum _Ecore_Exe_Flags

**Enumerator:**

***ECORE_EXE_PIPE_READ*** Exe Pipe Read mask.

***ECORE_EXE_PIPE_WRITE*** Exe Pipe Write mask.

***ECORE_EXE_PIPE_ERROR*** Exe Pipe error mask.

***ECORE_EXE_PIPE_READ_LINE_BUFFERED*** Reads are buffered until a newline and delivered 1 event per line.

***ECORE_EXE_PIPE_ERROR_LINE_BUFFERED*** Errors are buffered until a newline and delivered 1 event per line.

***ECORE_EXE_PIPE_AUTO*** stdout and stderr are buffered automatically

***ECORE_EXE_RESPAWN*** FIXME: Exe is restarted if it dies.

***ECORE_EXE_USE_SH*** Use /bin/sh to run the command.

#### 8.1.2.2 enum _Ecore_Fd_Handler_Flags

**Enumerator:**

***ECORE_FD_READ*** Fd Read mask.

***ECORE_FD_WRITE*** Fd Write mask.

***ECORE_FD_ERROR*** Fd Error mask.

### 8.1.3 Function Documentation

#### 8.1.3.1 EAPI Ecore_Animator* ecore_animator_add (int(*)(void *data) *func*, const void * *data*)

Add a animator to tick off at every animaton tick during main loop execution.

**Parameters:**

***func*** The function to call when it ticks off

***data*** The data to pass to the function

**Returns:**

A handle to the new animator

This function adds a animator and returns its handle on success and NULL on failure. The function `func` will be called every N seconds where N is the frametime interval set by ecore_animator_frametime_set(). The function will be passed the `data` pointer as its parameter.

When the animator `func` is called, it must return a value of either 1 or 0. If it returns 1, it will be called again at the next tick, or if it returns 0 it will be deleted automatically making any references/handles for it invalid.

#### 8.1.3.2 EAPI void* ecore_animator_del (Ecore_Animator * *animator*)

Delete the specified animator from the animator list.

**Parameters:**

***animator*** The animator to delete

**Returns:**

The data pointer set for the animator

Delete the specified `aqnimator` from the set of animators that are executed during main loop execution. This function returns the data parameter that was being passed to the callback on success, or NULL on failure. After this call returns the specified animator object `animator` is invalid and should not be used again. It will not get called again after deletion.

#### 8.1.3.3 EAPI double ecore_animator_frametime_get (void)

Get the animator call interval in seconds.

**Returns:**

The time in second in between animator ticks.

this function retrieves the time inbetween animator ticks, in seconds.

#### 8.1.3.4 EAPI void ecore_animator_frametime_set (double *frametime*)

Set the animator call interval in seconds.

**Parameters:**

***frametime*** The time in seconds in between animator ticks.

This function sets the time interval (in seconds) inbetween animator ticks.

#### 8.1.3.5 EAPI void ecore_app_args_get (int * *argc*, char *** *argv*)

Return the programs stored command-line arguments.

**Parameters:**

***argc*** A pointer to the return value to hold argc

***argv*** A pointer to the return value to hold argv

When called, this funciton returns the arguments for the program stored by ecore_app_args_set(). The integer pointed to by `argc` will be filled, if the pointer is not NULL, and the string array pointer `argv` will be filled also if the pointer is not NULL. The values they are filled with will be the same set by ecore_app_args_set().

#### 8.1.3.6 EAPI void ecore_app_args_set (int *argc*, const char ** *argv*)

Set up the programs command-line arguments.

**Parameters:**

***argc*** The same as passed as argc to the programs main() function

***argv*** The same as passed as argv to the programs main() function

A call to this function will store the programs command-line arguments for later use by ecore_app_restart() or ecore_app_args_get().

#### 8.1.3.7 EAPI void ecore_app_restart (void)

Restart the program executable with the command-line arguments stored.

This function will restart & re-execute this program in place of itself using the command-line arguments stored by ecore_app_args_set(). This is an easy way for a program to restart itself for cleanup purposes, configuration reasons or in the event of a crash.

#### 8.1.3.8 EAPI Ecore_Event* ecore_event_add (int *type*, void * *ev*, void(*)(void *data, void *ev) *func_free*, void * *data*)

Add an event to the event queue.

**Parameters:**

***type*** The event type to add to the end of the event queue

***ev*** The private data structure for this event type

***func_free*** The function to be called to free this private structure

***data*** The data pointer to be passed to the free function

**Returns:**

A Handle for that event

On success this function returns a handle to an event on the event queue, or NULL if it fails. If it succeeds, an event of type `type` will be added to the queue for processing by event handlers added by ecore_event_handler_add(). The `ev` parameter will be a pointer to the event private data that is specific to that event type. When the event is no longer needed, `func_free` will be called and passed the private sructure pointer for cleaning up. If `func_free` is NULL, free() will be called with the private structure pointer. func_free is passed `data` as its data parameter.

#### 8.1.3.9 EAPI void* ecore_event_current_event_get (void)

Return the current event type pointer handled.

**Returns:**

The current event pointer being handled if inside a handler callback

If the program is currently inside an Ecore event handler callback this will return the pointer of the current event being processed. If Ecore is not inside an event handler, NULL will be returned.

This is useful when certain Ecore modules such as Ecore_Evas "swallow" events and not all the original information is passed on. In special cases this extra information may be useful or needed and using this call can let the program access the event data if the type of the event is handled by the program.

#### 8.1.3.10 EAPI int ecore_event_current_type_get (void)

Return the current event type being handled.

**Returns:**

The current event type being handled if inside a handler callback

If the program is currently inside an Ecore event handler callback this will return the type of the current event being processed. If Ecore is not inside an event handler, ECORE_EVENT_NONE is returned.

This is useful when certain Ecore modules such as Ecore_Evas "swallow" events and not all the original information is passed on. In special cases this extra information may be useful or needed and using this call can let the program know if the event type being handled is one it wants to get more information about.

#### 8.1.3.11 EAPI void* ecore_event_del (Ecore_Event * *event*)

Delete an event from the queue.

**Parameters:**

***event*** The event handle to delete

**Returns:**

The data pointer originally set for the event free function

This deletes the event `event` from the event queue, and returns the `data` parameer originally set when adding it with ecore_event_add(). This does not immediately call the free function, and it may be called later on cleanup, and so if the free function depends on the data pointer to work, you should defer cleaning of this till the free function is called later.

#### 8.1.3.12 EAPI Ecore_Event_Filter* ecore_event_filter_add (void *(*)(void *data) *func_start*, int(*)(void *data, void *loop_data, int type, void *event) *func_filter*, void(*)(void *data, void *loop_data) *func_end*, const void * *data*)

Add a filter the current event queue.

**Parameters:**

***func_start*** Function to call just before filtering and return data

***func_filter*** Function to call on each event

***func_end*** Function to call after the queu has been filtered

***data*** Data to pass to the filter functions

**Returns:**

A filter handle

This adds a filter to call callbacks to loop through the event queue and filter events out of the queue. On failure NULL is returned. On success a Filter handle is returned. Filters are called on the queue just before Event handler processing to try and remove redundant events. Just as processing starts `func_start` is called and passed the `data` pointer. This function returns a pointer that is used as loop_data that is now passed to `func_filter` as loop_data. `func_filter` is also passed `data` and the event type and private event structure. If this callback returns 0, the event is removed from the queue. If it returns 1, the event is kept. When processing is finished `func_end` is called and is passed the loop_data and `data` pointer to clean up.

#### 8.1.3.13 EAPI void* ecore_event_filter_del (Ecore_Event_Filter * *ef*)

Delete an event filter.

**Parameters:**

***ef*** The event filter handle

**Returns:**

The data set for the filter

Delete a filter that has been added by its `ef` handle. On success this will return the data pointer set when this filter was added. On failure NULL is returned.

#### 8.1.3.14 EAPI Ecore_Event_Handler* ecore_event_handler_add (int *type*, int(*)(void *data, int type, void *event) *func*, const void * *data*)

Add an event handler.

**Parameters:**

***type*** The type of the event this handler will get called for

***func*** The function to call when the event is found in the queue

***data*** A data pointer to pass to the called function `func`

**Returns:**

A new Event handler, or NULL on failure

Add an event handler to the list of handlers. This will, on success, return a handle to the event handler object that was created, that can be used later to remove the handler using ecore_event_handler_del(). The `type` parameter is the iteger of the event type that will trigger this callback to be called. The callback `func` is called when this event is processed and will be passed the event type, a pointer to the private event structure that is specific to that event type, and a data pointer that is provided in this call as the `data` parameter.

When the callback `func` is called, it must return 1 or 0. If it returns 1, It will keep being called as per normal, for each handler set up for that event type. If it returns 0, it will cease processing handlers for that particular event, so all handler set to handle that event type that have not already been called, will not be.

#### 8.1.3.15 EAPI void* ecore_event_handler_del (Ecore_Event_Handler * *event_handler*)

Delete an event handler.

**Parameters:**

***event_handler*** Event handler handle to delete

**Returns:**

Data passed to handler

Delete a specified event handler from the handler list. On success this will delete the event handler and return the pointer passed as `data` when the handler was added by ecore_event_handler_add(). On failure NULL will be returned. Once a handler is deleted it will no longer be called.

#### 8.1.3.16 EAPI int ecore_event_type_new (void)

Allocate a new event type id sensibly and return the new id.

**Returns:**

A new event type id.

This function allocates a new event type id and returns it. Once an event type has been allocated it can never be de-allocated during the life of the program. There is no guarantee of the contents of this event ID, or how it is calculated, except that the ID will be unique to the current instance of the process.

#### 8.1.3.17 EAPI int ecore_init (void)

Set up connections, signal handlers, sockets etc.

**Returns:**

1 or greater on success, 0 otherwise

This function sets up all singal handlers and the basic event loop. If it succeeds, 1 will be returned, otherwise 0 will be returned.

```
#include <Ecore.h>

int main(int argc, char **argv)
{
  if (!ecore_init())
  {
    printf("ERROR: Cannot init Ecore!\n");
    return -1;
  }
  ecore_main_loop_begin();
  ecore_shutdown();
}
```

#### 8.1.3.18 EAPI int ecore_shutdown (void)

Shut down connections, signal handlers sockets etc.

This function shuts down all things set up in ecore_init() and cleans up all event queues, handlers, filters, timers, idlers, idle enterers/exiters etc. set up after ecore_init() was called.

Do not call this function from any callback that may be called from the main loop, as the main loop will then fall over and not function properly.

## 8.2 Ecore_Con.h File Reference

Sockets functions.

### Typedefs

- typedef _Ecore_Con_Server Ecore_Con_Server
  *A connection handle.*
- typedef _Ecore_Con_Client Ecore_Con_Client
  *A connection handle.*

### Functions

- EAPI int ecore_con_init (void)
  *Initialises the Ecore_ Con library.*
- EAPI int ecore_con_shutdown (void)
  *Shuts down the Ecore_ Con library.*
- EAPI Ecore_Con_Server * ecore_con_server_add (Ecore_Con_Type type, const char *name, int port, const void *data)
  *Creates a server to listen for connections.*
- EAPI Ecore_Con_Server * ecore_con_server_connect (Ecore_Con_Type type, const char *name, int port, const void *data)
  *Creates a server object to represent the server listening at the given port.*
- EAPI void * ecore_con_server_del (Ecore_Con_Server *svr)
  *Closes the connection and frees the given server.*
- EAPI void * ecore_con_server_data_get (Ecore_Con_Server *svr)
  *Retrieves the data associated with the given server.*
- EAPI int ecore_con_server_connected_get (Ecore_Con_Server *svr)
  *Retrieves whether the given server is currently connected.*
- EAPI Ecore_List * ecore_con_server_clients_get (Ecore_Con_Server *svr)
  *Retrieves the current list of clients.*
- EAPI int ecore_con_server_send (Ecore_Con_Server *svr, const void *data, int size)
  *Sends the given data to the given server.*
- EAPI void ecore_con_server_client_limit_set (Ecore_Con_Server *svr, int client_limit, char reject_excess_clients)

*Sets a limit on the number of clients that can be handled concurrently by the given server, and a policy on what to do if excess clients try to connect.*

- EAPI char * ecore_con_server_ip_get (Ecore_Con_Server *svr)

  *Gets the IP address of a server that has been connected to.*

- EAPI void ecore_con_server_flush (Ecore_Con_Server *svr)

  *Flushes all pending data to the given server.*

- EAPI int ecore_con_client_send (Ecore_Con_Client *cl, void *data, int size)

  *Sends the given data to the given client.*

- EAPI Ecore_Con_Server * ecore_con_client_server_get (Ecore_Con_Client *cl)

  *Retrieves the server representing the socket the client has connected to.*

- EAPI void * ecore_con_client_del (Ecore_Con_Client *cl)

  *Closes the connection and frees memory allocated to the given client.*

- EAPI void ecore_con_client_data_set (Ecore_Con_Client *cl, const void *data)

  *Sets the data associated with the given client to* `data`.

- EAPI void * ecore_con_client_data_get (Ecore_Con_Client *cl)

  *Retrieves the data associated with the given client.*

- EAPI char * ecore_con_client_ip_get (Ecore_Con_Client *cl)

  *Gets the IP address of a cleint that has connected.*

- EAPI void ecore_con_client_flush (Ecore_Con_Client *cl)

  *Flushes all pending data to the given client.*

- EAPI int ecore_con_ssl_available_get (void)

  *Returns if SSL support is available.*

### 8.2.1 Detailed Description

Sockets functions.

The Ecore Connection Library ( `Ecore_Con` ) provides simple mechanisms for communications between programs using reliable sockets. It saves the programmer from having to worry about file descripters and waiting for incoming connections.

There are two main objects in the `Ecore_Con` library: the `Ecore_Con_Server` and the `Ecore_Con_Client`.

The `Ecore_Con_Server` represents a server to connect to. It is represents a server that can be connected to. It is used regardless of whether the program is acting as a server or client itself.

To create a listening server, call `ecore_con_server_add()`.

To connect to a server, call `ecore_con_server_connect()`. Data can then be sent to the server using the `ecore_con_server_send()`.

Whenever a client connection is made to an `Ecore_Con_Server`, a `ECORE_CON_CLIENT_ADD` event is emitted. Any event callbacks that are called receive a `Ecore_Con_Client` object, which represents a connection that that particular client.

Functions are described in the following groupings:

- Ecore Connection Library Functions
- Ecore Connection Server Functions
- Ecore Connection Client Functions

## 8.3 Ecore_Config.h File Reference

Provides the Enlightened Property Library.

### Data Structures

- struct Ecore_Config_Prop

  *The actual property for storing a key-value pair.*

### Typedefs

- typedef int(*) Ecore_Config_Listener (const char *key, const Ecore_Config_Type type, const int tag, void *data)

  *Property change callback function prototype.*

### Enumerations

- enum Ecore_Config_Type {
  ECORE_CONFIG_NIL = 0,
  ECORE_CONFIG_INT = 1,
  ECORE_CONFIG_FLT = 2,
  ECORE_CONFIG_STR = 3,
  ECORE_CONFIG_RGB = 4,
  ECORE_CONFIG_THM = 5,
  ECORE_CONFIG_BLN = 6,
  ECORE_CONFIG_SCT = 7 }

  *Valid configuration property types.*

### Functions

- EAPI Ecore_Config_Prop * ecore_config_get (const char *key)

  *Returns the property with the given key.*

- EAPI const char * ecore_config_type_get (const Ecore_Config_Prop *e)

  *Returns the type of the property.*

- EAPI int ecore_config_boolean_get (const char *key)

  *Returns the specified property as an integer.*

- EAPI char * ecore_config_string_get (const char *key)

*Returns the specified property as a string.*

- EAPI long ecore_config_int_get (const char *key)
  *Returns the specified property as a long integer.*
- EAPI int ecore_config_argb_get (const char *key, int *a, int *r, int *g, int *b)
  *Finds the alpha, red, green and blue values of a color property.*
- EAPI long ecore_config_argbint_get (const char *key)
  *Returns a color property as a long.*
- EAPI char * ecore_config_argbstr_get (const char *key)
  *Returns a color property as a string of hexadecimal characters.*
- EAPI float ecore_config_float_get (const char *key)
  *Returns the specified property as a float.*
- EAPI char * ecore_config_theme_get (const char *key)
  *Returns a theme property.*
- EAPI char * ecore_config_as_string_get (const char *key)
  *Retrieves the key as a string.*
- EAPI int ecore_config_describe (const char *key, const char *desc)
  *Sets the description field of the indicated property.*
- EAPI int ecore_config_short_opt_set (const char *key, char short_opt)
  *Set the short option character of a property.*
- EAPI int ecore_config_long_opt_set (const char *key, const char *long_opt)
  *Set the long option string of the property.*
- EAPI int ecore_config_set (const char *key, const char *val)
  *Sets the indicated property to the value indicated by* val.
- EAPI int ecore_config_typed_set (const char *key, const void *val, int type)
  *Sets the indicated property to the given value and type.*
- EAPI int ecore_config_boolean_set (const char *key, int val)
  *Sets the indicated property to the given boolean.*
- EAPI int ecore_config_string_set (const char *key, const char *val)
  *Sets the indicated property to the given string.*
- EAPI int ecore_config_int_set (const char *key, int val)
  *Sets the indicated property to the given integer.*
- EAPI int ecore_config_argb_set (const char *key, int a, int r, int g, int b)
  *Sets the indicated property to a color value.*

- EAPI int ecore_config_argbint_set (const char *key, long argb)
  *Sets the indicated property to a color value.*
- EAPI int ecore_config_argbstr_set (const char *key, const char *val)
  *Sets the indicated property to a color value.*
- EAPI int ecore_config_float_set (const char *key, float val)
  *Sets the indicated property to the given float value.*
- EAPI int ecore_config_theme_set (const char *key, const char *val)
  *Sets the indicated property to a theme name.*
- EAPI int ecore_config_theme_preview_group_set (const char *key, const char *group)
  *Sets the theme preview group of an indicated property.*
- EAPI int ecore_config_as_string_set (const char *key, const char *val)
  *Sets the indicated property to the value given in the string.*
- EAPI int ecore_config_default (const char *key, const char *val, float lo, float hi, float step)
  *Sets the indicated property if it has not already been set or loaded.*
- EAPI int ecore_config_boolean_default (const char *key, int val)
  *Sets the indicated property to the given boolean if the property has not yet been set.*
- EAPI int ecore_config_int_default (const char *key, int val)
  *Sets the indicated property to the given integer if the property has not yet been set.*
- EAPI int ecore_config_int_default_bound (const char *key, int val, int lo, int hi, int step)
  *Sets the indicated property to the given integer if the property has not yet been set.*
- EAPI int ecore_config_string_default (const char *key, const char *val)
  *Sets the indicated property to the given string if the property has not yet been set.*
- EAPI int ecore_config_float_default (const char *key, float val)
  *Sets the indicated property to the given float if the property has not yet been set.*
- EAPI int ecore_config_float_default_bound (const char *key, float val, float lo, float hi, float step)
  *Sets the indicated property to the given float if the property has not yet been set.*
- EAPI int ecore_config_argb_default (const char *key, int a, int r, int g, int b)
  *Sets the indicated property to a color value if the property has not yet been set.*
- EAPI int ecore_config_argbint_default (const char *key, long argb)
  *Sets the indicated property to a color value if the property has not yet been set.*
- EAPI int ecore_config_argbstr_default (const char *key, const char *val)
  *Sets the indicated property to a color value if the property has not yet been set.*

- EAPI int ecore_config_theme_default (const char *key, const char *val)

  *Sets the indicated property to a theme name if the property has not yet been set.*

- EAPI int ecore_config_struct_int_add (const char *key, const char *name, int val)

  *Add an int property to the named structure.*

- EAPI int ecore_config_struct_float_add (const char *key, const char *name, float val)

  *Add a float property to the named structure.*

- EAPI int ecore_config_struct_create (const char *key)

  *Sets the indicated property to a structure if the property has not yet been set.*

- EAPI int ecore_config_struct_string_add (const char *key, const char *name, const char *val)

  *Add a string property to the named structure.*

- EAPI int ecore_config_struct_theme_add (const char *key, const char *name, const char *val)

  *Add a theme property to the named structure.*

- EAPI int ecore_config_struct_argb_add (const char *key, const char *name, int a, int r, int g, int b)

  *Add an argb property to the named structure.*

- EAPI int ecore_config_struct_boolean_add (const char *key, const char *name, int val)

  *Add a boolean property to the named structure.*

- EAPI int ecore_config_struct_get (const char *key, void *data)

  *Get the contents of a defined structure property and load it into the passed C struct.*

- EAPI int ecore_config_listen (const char *name, const char *key, Ecore_Config_Listener listener, int tag, void *data)

  *Adds a callback function to the list of functions called when a property changes.*

- EAPI int ecore_config_deaf (const char *name, const char *key, Ecore_Config_Listener listener)

  *Removes a listener callback.*

- EAPI Ecore_Config_Prop * ecore_config_dst (Ecore_Config_Prop *e)

  *Removes the given property from the local configuration and destroys it.*

- EAPI int ecore_config_type_guess (const char *key, const char *val)

  *Tries to guess the type of a property.*

- EAPI Ecore_Config_Bundle * ecore_config_bundle_new (Ecore_Config_Server *srv, const char *id)

  *Creates a new Ecore_Config_Bundle.*

- EAPI Ecore_Config_Bundle * ecore_config_bundle_1st_get (Ecore_Config_Server *srv)

*Locates the first configuration bundle on the given server.*

- EAPI Ecore_Config_Bundle * ecore_config_bundle_next_get (Ecore_Config_Bundle *ns)

  *Locates the configuration bundle after the given one.*

- EAPI Ecore_Config_Bundle * ecore_config_bundle_by_serial_get (Ecore_Config_Server *srv, long serial)

  *Locates a configuration bundle on a configuration server based on its serial number.*

- EAPI Ecore_Config_Bundle * ecore_config_bundle_by_label_get (Ecore_Config_Server *srv, const char *label)

  *Gets the Ecore_ Config_ Bundle with the given identifier from the given server.*

- EAPI long ecore_config_bundle_serial_get (Ecore_Config_Bundle *ns)

  *Retrieves the bundle's serial number.*

- EAPI char * ecore_config_bundle_label_get (Ecore_Config_Bundle *ns)

  *Retrieves the bundle's identifier.*

- EAPI int ecore_config_init (const char *name)

  *Initializes the Enlightened Property Library.*

- EAPI int ecore_config_shutdown (void)

  *Frees memory and shuts down the library for an application.*

- EAPI int ecore_config_system_init (void)

  *Initializes the Enlightened Property Library.*

- EAPI int ecore_config_system_shutdown (void)

  *Frees memory and shuts down the library for other programming libraries.*

- EAPI int ecore_config_load (void)

  *Loads the default configuration.*

- EAPI int ecore_config_file_load (const char *file)

  *Load the given configuration file to the local configuration.*

- EAPI int ecore_config_save (void)

  *Saves the current configuration to the default file.*

- EAPI int ecore_config_file_save (const char *file)

  *Saves the local configuration to the given file.*

- EAPI int ecore_config_evas_font_path_apply (Evas *evas)

  *Calls evas_font_path_ append on* `evas` *for each of the font names stored in the property "/e/font/path".*

- EAPI char * ecore_config_theme_search_path_get (void)

  *Retrieves the search path used to find themes.*

- EAPI int ecore_config_theme_search_path_append (char *append)

  *Adds the given path to the search path used to find themes.*

- EAPI char * ecore_config_theme_default_path_get (void)

  *Retrieves the default theme search path.*

- EAPI char * ecore_config_theme_with_path_from_name_get (char *name)

  *Retrieve a theme file's full path.*

- EAPI char * ecore_config_theme_with_path_get (const char *key)

  *Retrieves the full path to the theme file of the theme stored in the given property.*

- EAPI void ecore_config_args_display (void)

  *Prints the property list of the local configuration bundle to output.*

- EAPI int ecore_config_args_parse (void)

  *Parse the arguments set by ecore_app_args_set and set properties accordingly.*

- EAPI void ecore_config_app_describe (char *description)

  *Sets the description string used by ecore_config_args_display .*

- EAPI int ecore_config_create (const char *key, void *val, char short_opt, char *long_opt, char *desc)

  *Creates a new property, if it does not already exist, and sets its attributes to those given.*

- EAPI int ecore_config_typed_create (const char *key, void *val, int type, char short_opt, char *long_opt, char *desc)

  *Creates a new property, if it does not already exist, and sets its attributes to those given.*

- EAPI int ecore_config_boolean_create (const char *key, int val, char short_opt, char *long_opt, char *desc)

  *Creates a new boolean property, if it does not already exist, and sets its attributes to those given.*

- EAPI int ecore_config_int_create (const char *key, int val, char short_opt, char *long_opt, char *desc)

  *Creates a new integer property, if it does not already exist, and sets its attributes to those given.*

- EAPI int ecore_config_int_create_bound (const char *key, int val, int low, int high, int step, char short_opt, char *long_opt, char *desc)

  *Creates a new integer property, if it does not already exist, and sets its attributes to those given.*

- EAPI int ecore_config_string_create (const char *key, char *val, char short_opt, char *long_opt, char *desc)

  *Creates a new string property, if it does not already exist, and sets its attributes to those given.*

- EAPI int ecore_config_float_create (const char *key, float val, char short_opt, char *long_opt, char *desc)

  *Creates a new float property, if it does not already exist, and sets its attributes to those given.*

- EAPI int ecore_config_float_create_bound (const char *key, float val, float low, float high, float step, char short_opt, char *long_opt, char *desc)

  *Creates a new float property, if it does not already exist, and sets its attributes to those given.*

- EAPI int ecore_config_argb_create (const char *key, char *val, char short_opt, char *long_opt, char *desc)

  *Creates a new color property, if it does not already exist, and sets its attributes to those given.*

- EAPI int ecore_config_theme_create (const char *key, char *val, char short_opt, char *long_opt, char *desc)

  *Creates a new theme property, if it does not already exist, and sets its attributes to those given.*

### 8.3.1 Detailed Description

Provides the Enlightened Property Library.

This file provies all headers and structs for use with Ecore_Config. Using individual header files should not be necessary.

### 8.3.2 Enumeration Type Documentation

#### 8.3.2.1 enum Ecore_Config_Type

Valid configuration property types.

**Enumerator:**

***ECORE_CONFIG_NIL*** Property with no value.
***ECORE_CONFIG_INT*** Integer property type.
***ECORE_CONFIG_FLT*** Float property type.
***ECORE_CONFIG_STR*** String property type.
***ECORE_CONFIG_RGB*** Colour property type.
***ECORE_CONFIG_THM*** Theme property type.
***ECORE_CONFIG_BLN*** Boolean property type.
***ECORE_CONFIG_SCT*** Structure property type.

### 8.3.3 Function Documentation

#### 8.3.3.1 EAPI void ecore_config_app_describe (char * *description*)

Sets the description string used by ecore_config_args_display .

**Parameters:**

***description*** Description of application.

#### 8.3.3.2 EAPI int ecore_config_args_parse (void)

Parse the arguments set by ecore_app_args_set and set properties accordingly.

**Returns:**

`ECORE_CONFIG_PARSE_CONTINUE` if successful. `ECORE_CONFIG_PARSE_EXIT` is returned if an unrecognised option is found. `ECORE_CONFIG_PARSE_HELP` is returned if help was displayed.

#### 8.3.3.3 EAPI Ecore_Config_Bundle* ecore_config_bundle_1st_get (Ecore_Config_Server * *srv*)

Locates the first configuration bundle on the given server.

**Parameters:**

***srv*** The configuration server.

**Returns:**

Pointer to the first configuration bundle.

#### 8.3.3.4 EAPI Ecore_Config_Bundle* ecore_config_bundle_by_label_get (Ecore_Config_Server * *srv*, const char * *label*)

Gets the Ecore_Config_Bundle with the given identifier from the given server.

**Parameters:**

***srv*** The configuration server.

***label*** The bundle's identifier string.

**Returns:**

The bundle with the given identifier string, or `NULL` if it could not be found.

#### 8.3.3.5 EAPI Ecore_Config_Bundle* ecore_config_bundle_by_serial_get (Ecore_Config_Server * *srv*, long *serial*)

Locates a configuration bundle on a configuration server based on its serial number.

**Parameters:**

***srv*** The configuration server.

***serial*** Serial number.

**Returns:**

The configuration bundle with the given serial number.

#### 8.3.3.6 EAPI char* ecore_config_bundle_label_get (Ecore_Config_Bundle * *ns*)

Retrieves the bundle's identifier.

**Parameters:**

***ns*** The configuration bundle.

**Returns:**

The bundle's identifer string.

#### 8.3.3.7 EAPI Ecore_Config_Bundle* ecore_config_bundle_new (Ecore_Config_Server * *srv*, const char * *identifier*)

Creates a new Ecore_Config_Bundle.

**Parameters:**

***srv*** Config server.

***identifier*** Identifier string for the new bundle.

**Returns:**

A pointer to a new Ecore_Config_Bundle. `NULL` is returned if the structure couldn't be allocated.

#### 8.3.3.8 EAPI Ecore_Config_Bundle* ecore_config_bundle_next_get (Ecore_Config_Bundle * *ns*)

Locates the configuration bundle after the given one.

**Parameters:**

***ns*** The configuration bundle.

**Returns:**

The next configuration bundle.

#### 8.3.3.9 EAPI long ecore_config_bundle_serial_get (Ecore_Config_Bundle * *ns*)

Retrieves the bundle's serial number.

**Parameters:**

***ns*** The configuration bundle.

**Returns:**

The bundle's identifier string, or -1 if ns is `NULL`.

#### 8.3.3.10 EAPI int ecore_config_evas_font_path_apply (Evas * *evas*)

Calls evas_font_path_append on `evas` for each of the font names stored in the property "/e/font/path".

**Parameters:**

***evas*** Evas object to append the font names to.

**Returns:**

`ECORE_CONFIG_ERR_SUCC` on success. `ECORE_CONFIG_ERR_NODATA` is returned if the property has not been set.

#### 8.3.3.11 EAPI char* ecore_config_theme_default_path_get (void)

Retrieves the default theme search path.

**Returns:**

The default theme search path.

#### 8.3.3.12 EAPI int ecore_config_theme_search_path_append (char * *path*)

Adds the given path to the search path used to find themes.

If the search path is successfully, the new search path will be saved into the property "/e/themes/search_path". Therefore, this function should be called **after** ecore_config_load to allow a user to override the default search path.

**Parameters:**

***path*** The given

**Returns:**

`ECORE_CONFIG_ERR_SUCC` on success. `ECORE_CONFIG_ERR_FAIL` will be returned if `path` already exists in the search path. `ECORE_CONFIG_ERR_FAIL` is returned if `path` is `NULL`.

#### 8.3.3.13 EAPI char* ecore_config_theme_search_path_get (void)

Retrieves the search path used to find themes.

The search path is stored in the property "/e/themes/search_path". If the property has not been set, the default path used is "/usr/local/share/<app_name>/themes|~/.e/apps/<app_name>/themes". See ecore_config_theme_default_path_get for more information about the default path.

**Returns:**

The search path. `NULL` is returned if there is no memory left.

#### 8.3.3.14 EAPI char* ecore_config_theme_with_path_from_name_get (char * *name*)

Retrieve a theme file's full path.

The search path for theme files is given by ecore_config_theme_search_path_get .

**Parameters:**

***name*** The name of the theme.

**Returns:**

A full path to the theme on success. `NULL` will be returned if `name` is `NULL` or no theme matching the given name could be found.

#### 8.3.3.15 EAPI char* ecore_config_theme_with_path_get (const char * *key*)

Retrieves the full path to the theme file of the theme stored in the given property.

The search path for themes is given by ecore_config_theme_search_path_get .

**Parameters:**

***key*** The given property.

**Returns:**

A full path to the theme on success, or `NULL` on failure. This function will fail if no key is specified or not theme matching that given by the property `key` could be found.

#### 8.3.3.16 EAPI int ecore_config_type_guess (const char * *key*, const char * *val*)

Tries to guess the type of a property.

This function first checks to see if the property exists. If it does, then the type of the stored property is returned. Otherwise, the function tries to guess the type of the property based on `val`.

**Parameters:**

***key*** The property key.

***val*** The value in string form.

**Returns:**

The type of the property determined by the function. Note that if val is `NULL`, `ECORE_CONFIG_NIL` will be returned.

## 8.4 Ecore_Data.h File Reference

Contains threading, list, hash, debugging and tree functions.

### Functions

- EAPI int ecore_direct_compare (const void *key1, const void *key2)
  *Perform a direct comparison of two keys' values.*
- EAPI int ecore_str_compare (const void *key1, const void *key2)
  *Perform a string comparison of two keys values.*
- EAPI unsigned int ecore_direct_hash (const void *key)
  *Just casts the key to an unsigned int.*
- EAPI unsigned int ecore_str_hash (const void *key)
  *Compute the hash value of a string.*
- EAPI Ecore_List * ecore_list_new (void)
  *Create and initialize a new list.*
- EAPI int ecore_list_init (Ecore_List *list)
  *Initialize a list to some sane starting values.*
- EAPI int ecore_list_append (Ecore_List *list, void *_data)
  *Append data to the list.*
- EAPI int ecore_list_prepend (Ecore_List *list, void *_data)
  *Prepend data to the beginning of the list.*
- EAPI int ecore_list_insert (Ecore_List *list, void *_data)
  *Insert data in front of the current point in the list.*
- EAPI int ecore_list_append_list (Ecore_List *list, Ecore_List *append)
  *Append a list to the list.*
- EAPI int ecore_list_prepend_list (Ecore_List *list, Ecore_List *prepend)
  *Prepend a list to the beginning of the list.*
- EAPI int ecore_list_remove_destroy (Ecore_List *list)
  *Remove and free the data in lists current position.*
- EAPI void * ecore_list_remove (Ecore_List *list)
  *Remove the current item from the list.*
- EAPI void * ecore_list_remove_first (Ecore_List *list)

*Remove the first item from the list.*

- EAPI void * ecore_list_remove_last (Ecore_List *list)

  *Remove the last item from the list.*

- EAPI void * ecore_list_current (Ecore_List *list)

  *Retrieve the data pointed to by the current item in* `list`.

- EAPI void * ecore_list_first (Ecore_List *list)

  *Retrieve the data pointed to by the first item in* `list`.

- EAPI void * ecore_list_last (Ecore_List *list)

  *Retrieve the data pointed to by the last item in* `list`.

- EAPI int ecore_list_index (Ecore_List *list)

  *Returns the number of the current node.*

- EAPI int ecore_list_nodes (Ecore_List *list)

  *Find the number of nodes in the list.*

- EAPI int ecore_list_for_each (Ecore_List *list, Ecore_For_Each function, void *user_data)

  *Execute function for each node in* `list`.

- EAPI void * ecore_list_goto_first (Ecore_List *list)

  *Make the current item the first item in the list.*

- EAPI void * ecore_list_goto_last (Ecore_List *list)

  *Make the current item the last item in the list.*

- EAPI void * ecore_list_goto_index (Ecore_List *list, int index)

  *Make the current item the item with the given index number.*

- EAPI void * ecore_list_goto (Ecore_List *list, void *_data)

  *Make the current item the node that contains* `data`.

- EAPI void * ecore_list_next (Ecore_List *list)

  *Retrieve the data pointed to by the current item, and make the next item the current item.*

- EAPI void * ecore_list_find (Ecore_List *list, Ecore_Compare_Cb function, const void *user_data)

  *Find data in* `list` *using the compare function* `func`.

- EAPI int ecore_list_sort (Ecore_List *list, Ecore_Compare_Cb compare, char order)

  *Sort data in* `list` *using the compare function* `compare`.

- EAPI int ecore_list_mergesort (Ecore_List *list, Ecore_Compare_Cb compare, char order)

  *Sort data in* `list` *using the compare function* `compare`.

- EAPI int ecore_list_heapsort (Ecore_List *list, Ecore_Compare_Cb compare, char order)

  *Sort data in* `list` *using the compare function* `compare`.

- EAPI int ecore_list_is_empty (Ecore_List *list)

  *Checks the list for any nodes.*

- EAPI int ecore_list_clear (Ecore_List *list)

  *Remove all nodes from* `list`.

- EAPI void ecore_list_destroy (Ecore_List *list)

  *Free a list and all of it's nodes.*

- EAPI Ecore_List_Node * ecore_list_node_new (void)

  *Allocates and initializes a new list node.*

- EAPI int ecore_list_node_destroy (Ecore_List_Node *_e_node, Ecore_Free_Cb free_func)

  *Calls the function to free the data and the node.*

- EAPI int ecore_list_set_free_cb (Ecore_List *list, Ecore_Free_Cb free_func)

  *Set the function for freeing data.*

- EAPI Ecore_DList * ecore_dlist_new (void)

  *Creates and initialises a new doubly linked list.*

- EAPI int ecore_dlist_init (Ecore_DList *list)

  *Initialises a list to some sane starting values.*

- EAPI void ecore_dlist_destroy (Ecore_DList *list)

  *Frees a doubly linked list and all of its nodes.*

- EAPI int ecore_dlist_append (Ecore_DList *_e_dlist, void *_data)

  *Appends data to the given doubly linked list.*

- EAPI int ecore_dlist_prepend (Ecore_DList *_e_dlist, void *_data)

  *Adds data to the very beginning of the given doubly linked list.*

- EAPI int ecore_dlist_insert (Ecore_DList *_e_dlist, void *_data)

  *Inserts data at the current point in the given doubly linked list.*

- EAPI int ecore_dlist_append_list (Ecore_DList *_e_dlist, Ecore_DList *append)

  *Appends a list to the given doubly linked list.*

- EAPI int ecore_dlist_prepend_list (Ecore_DList *_e_dlist, Ecore_DList *prepend)

  *Adds a list to the very beginning of the given doubly linked list.*

- EAPI void * ecore_dlist_current (Ecore_DList *list)

  *Return the data in the current list item.*

- EAPI int ecore_dlist_index (Ecore_DList *list)

  *Retrieves the index of the current node of the given doubly linked list.*

- EAPI void * ecore_dlist_remove (Ecore_DList *_e_dlist)

  *Removes the current item from the given doubly linked list.*

- EAPI void * ecore_dlist_remove_first (Ecore_DList *_e_dlist)

  *Removes the first item from the given doubly linked list.*

- EAPI int ecore_dlist_remove_destroy (Ecore_DList *list)

  *Removes and frees the data at the current position in the given doubly linked list.*

- EAPI void * ecore_dlist_remove_last (Ecore_DList *_e_dlist)

  *Removes the last item from the given doubly linked list.*

- EAPI void * ecore_dlist_goto_first (Ecore_DList *_e_dlist)

  *Move the current pointer to the first item in the list.*

- EAPI void * ecore_dlist_goto_last (Ecore_DList *_e_dlist)

  *Move the pointer to the current item to the last item.*

- EAPI void * ecore_dlist_goto_index (Ecore_DList *_e_dlist, int index)

  *Moves the current item to the index number in the given doubly linked list.*

- EAPI void * ecore_dlist_goto (Ecore_DList *_e_dlist, void *_data)

  *Move the current item to the node that contains data.*

- EAPI void * ecore_dlist_next (Ecore_DList *list)

  *Move to the next item in the list and return current item.*

- EAPI void * ecore_dlist_previous (Ecore_DList *list)

  *Move to the previous item and return current item.*

- EAPI int ecore_dlist_sort (Ecore_DList *list, Ecore_Compare_Cb compare, char order)

  *Sort data in* `list` *using the compare function* `compare`.

- EAPI int ecore_dlist_mergesort (Ecore_DList *list, Ecore_Compare_Cb compare, char order)

  *Sort data in* `list` *using the compare function* `compare`.

- EAPI int ecore_dlist_is_empty (Ecore_DList *_e_dlist)

  *Returns whether there is anything in the given doubly linked list.*

- EAPI int ecore_dlist_clear (Ecore_DList *_e_dlist)

  *Remove all nodes from the list.*

- EAPI int ecore_dlist_set_free_cb (Ecore_DList *dlist, Ecore_Free_Cb free_func)

  *Sets the function used for freeing data stored in a doubly linked list.*

- EAPI Ecore_Hash * ecore_hash_new (Ecore_Hash_Cb hash_func, Ecore_Compare_Cb compare)

  *Creates and initializes a new hash.*

- EAPI int ecore_hash_init (Ecore_Hash *hash, Ecore_Hash_Cb hash_func, Ecore_Compare_Cb compare)

  *Initializes the given hash.*

- EAPI int ecore_hash_set_free_key (Ecore_Hash *hash, Ecore_Free_Cb function)

  *Sets the function to destroy the keys of the given hash.*

- EAPI int ecore_hash_set_free_value (Ecore_Hash *hash, Ecore_Free_Cb function)

  *Sets the function to destroy the values in the given hash.*

- EAPI void ecore_hash_destroy (Ecore_Hash *hash)

  *Frees the hash table and the data contained inside it.*

- EAPI int ecore_hash_count (Ecore_Hash *hash)

  *Counts the number of nodes in a hash table.*

- EAPI int ecore_hash_for_each_node (Ecore_Hash *hash, Ecore_For_Each for_each_func, void *user_data)

  *Runs the* `for_each_func` *function on each entry in the given hash.*

- EAPI Ecore_List * ecore_hash_keys (Ecore_Hash *hash)

  *Retrieves an ecore_list of all keys in the given hash.*

- EAPI void * ecore_hash_get (Ecore_Hash *hash, const void *key)

  *Retrieves the value associated with the given key from the given hash table.*

- EAPI int ecore_hash_set (Ecore_Hash *hash, void *key, void *value)

  *Sets a key-value pair in the given hash table.*

- EAPI int ecore_hash_set_hash (Ecore_Hash *hash, Ecore_Hash *set)

  *Sets all key-value pairs from set in the given hash table.*

- EAPI void * ecore_hash_remove (Ecore_Hash *hash, const void *key)

  *Removes the value associated with the given key in the given hash table.*

- EAPI void * ecore_hash_find (Ecore_Hash *hash, Ecore_Compare_Cb compare, const void *value)

  *Retrieves the first value that matches table.*

- EAPI void ecore_hash_dump_graph (Ecore_Hash *hash)

  *Prints the distribution of the given hash table for graphing.*

- EAPI void ecore_hash_dump_stats (Ecore_Hash *hash)

  *Prints the distribution of the given hash table for graphing.*

- EAPI int ecore_path_group_new (char *group_name)

*Creates a new path group.*

- EAPI void ecore_path_group_del (int group_id)
  *Destroys a previously created path group.*
- EAPI void ecore_path_group_add (int group_id, char *path)
  *Adds a directory to be searched for files.*
- EAPI void ecore_path_group_remove (int group_id, char *path)
  *Removes the given directory from the given group.*
- EAPI char * ecore_path_group_find (int group_id, char *name)
  *Finds a file in a group of paths.*
- EAPI Ecore_List * ecore_path_group_available (int group_id)
  *Retrieves a list of all available files in the given path.*
- EAPI Ecore_Plugin * ecore_plugin_load (int group_id, const char *plugin)
  *Loads the specified plugin from the specified path group.*
- EAPI void ecore_plugin_unload (Ecore_Plugin *plugin)
  *Unloads the given plugin from memory.*
- EAPI Ecore_Sheap * ecore_sheap_new (Ecore_Compare_Cb compare, int size)
  *Allocate and initialize a new binary heap.*
- EAPI void ecore_sheap_destroy (Ecore_Sheap *heap)
  *Free up the memory used by the heap.*
- EAPI int ecore_sheap_init (Ecore_Sheap *heap, Ecore_Compare_Cb compare, int size)
  *Initialize a binary heap to default values.*
- EAPI int ecore_sheap_set_free_cb (Ecore_Sheap *heap, Ecore_Free_Cb free_func)
  *Set the function for freeing data.*
- EAPI int ecore_sheap_insert (Ecore_Sheap *heap, void *data)
  *Insert new data into the heap.*
- EAPI void * ecore_sheap_extract (Ecore_Sheap *heap)
  *Extract the item at the top of the heap.*
- EAPI void * ecore_sheap_extreme (Ecore_Sheap *heap)
  *Examine the item at the top of the heap.*
- EAPI int ecore_sheap_change (Ecore_Sheap *heap, void *item, void *newval)
  *Change the value of the specified item in the heap.*
- EAPI int ecore_sheap_set_compare (Ecore_Sheap *heap, Ecore_Compare_Cb compare)
  *Change the comparison function for the heap.*

- EAPI void ecore_sheap_set_order (Ecore_Sheap *heap, char order)
  *Change the order of the heap.*
- EAPI void ecore_sheap_sort (Ecore_Sheap *heap)
  *Sort the data in the heap.*
- EAPI int ecore_string_init (void)
  *Initialize the ecore string internal structure.*
- EAPI void ecore_string_shutdown (void)
  *Shutdown the ecore string internal structures.*
- EAPI const char * ecore_string_instance (const char *string)
  *Retrieves an instance of a string for use in an ecore program.*
- EAPI void ecore_string_release (const char *string)
  *Notes that the given string has lost an instance.*
- EAPI Ecore_Tree * ecore_tree_new (Ecore_Compare_Cb compare_func)
  *Allocate a new tree structure.*
- EAPI int ecore_tree_init (Ecore_Tree *tree, Ecore_Compare_Cb compare_func)
  *Initialize a tree structure to some sane initial values.*
- EAPI int ecore_tree_destroy (Ecore_Tree *tree)
  *Free the tree and it's stored data.*
- EAPI int ecore_tree_is_empty (Ecore_Tree *tree)
  *Test to see if the tree has any nodes.*
- EAPI void * ecore_tree_get (Ecore_Tree *tree, void *key)
  *Return the value corresponding to key.*
- EAPI Ecore_Tree_Node * ecore_tree_get_node (Ecore_Tree *tree, void *key)
  *Return the node corresponding to key.*
- EAPI void * ecore_tree_get_closest_larger (Ecore_Tree *tree, void *key)
  *Find the closest value greater than or equal to the key.*
- EAPI void * ecore_tree_get_closest_smaller (Ecore_Tree *tree, void *key)
  *Find the closest value <= key.*
- EAPI int ecore_tree_set (Ecore_Tree *tree, void *key, void *value)
  *Set the value associated with key to* value.
- EAPI int ecore_tree_remove (Ecore_Tree *tree, void *key)
  *Remove the key from the tree.*
- EAPI int ecore_tree_add_node (Ecore_Tree *tree, Ecore_Tree_Node *node)

*Place a node in the tree.*

- EAPI int ecore_tree_remove_node (Ecore_Tree *tree, Ecore_Tree_Node *node)

  *Remove the node from the tree.*

- EAPI int ecore_tree_for_each_node (Ecore_Tree *tree, Ecore_For_Each for_each_func, void *user_data)

  *Execute the function for each node in the tree.*

- EAPI int ecore_tree_for_each_node_value (Ecore_Tree *tree, Ecore_For_Each for_each_func, void *user_data)

  *Execute function for each value in the tree.*

- EAPI Ecore_Strbuf * ecore_strbuf_new (void)

  *Create a new string buffer.*

- EAPI void ecore_strbuf_free (Ecore_Strbuf *buf)

  *Free a string buffer.*

- EAPI void ecore_strbuf_append (Ecore_Strbuf *buf, const char *str)

  *Append a string to a buffer, reallocating as necessary.*

- EAPI void ecore_strbuf_append_char (Ecore_Strbuf *buf, char c)

  *Append a character to a string buffer, reallocating as necessary.*

- EAPI void ecore_strbuf_insert (Ecore_Strbuf *buf, const char *str, size_t pos)

  *Insert a string to a buffer, reallocating as necessary.*

- EAPI const char * ecore_strbuf_string_get (Ecore_Strbuf *buf)

  *Retrieve a pointer to the contents of a string buffer.*

- EAPI size_t ecore_strbuf_length_get (Ecore_Strbuf *buf)

  *Retrieve the length of the string buffer content.*

- EAPI int ecore_strbuf_replace (Ecore_Strbuf *buf, const char *str, const char *with, unsigned int n)

  *Replace the n-th string with an other string.*

- EAPI int ecore_strbuf_replace_all (Ecore_Strbuf *buf, const char *str, const char *with)

  *Replace all strings with an other string.*

### 8.4.1 Detailed Description

Contains threading, list, hash, debugging and tree functions.

### 8.4.2 Function Documentation

#### 8.4.2.1 EAPI int ecore_direct_compare (const void * *key1*, const void * *key2*)

Perform a direct comparison of two keys' values.

**Parameters:**

***key1*** The first key to compare
***key2*** The second key to compare

**Returns:**

A strcmp style value to indicate the larger key

#### 8.4.2.2 EAPI unsigned int ecore_direct_hash (const void * *key*)

Just casts the key to an unsigned int.

**Parameters:**

***key*** The key to return compute a hash value

**Returns:**

The key cast to an unsigned int.

#### 8.4.2.3 EAPI int ecore_dlist_clear (Ecore_DList * *list*)

Remove all nodes from the list.

**Parameters:**

***list,:*** the list to remove all nodes from

**Returns:**

Returns TRUE on success, FALSE on errors

#### 8.4.2.4 EAPI void* ecore_dlist_current (Ecore_DList * *list*)

Return the data in the current list item.

**Parameters:**

***list,:*** the list to the return the current data

**Returns:**

Returns value of the current data item, NULL if no current item

#### 8.4.2.5 EAPI void* ecore_dlist_goto (Ecore_DList * *list*, void * *data*)

Move the current item to the node that contains data.

**Parameters:**

***list,:*** the list to move the current item in
***data,:*** the data to find and set the current item to

**Returns:**

Returns specified data on success, NULL on error

#### 8.4.2.6 EAPI void* ecore_dlist_goto_first (Ecore_DList * *list*)

Move the current pointer to the first item in the list.

**Parameters:**

***list,:*** the list to change the current to the first item

**Returns:**

Returns a pointer to the first item on success, NULL on failure.

#### 8.4.2.7 EAPI void* ecore_dlist_goto_index (Ecore_DList * *list*, int *index*)

Moves the current item to the index number in the given doubly linked list.

**Parameters:**

***list*** The given doubly linked list.
***index*** The position to move the current item

**Returns:**

The node at specified index on success, `NULL` on error.

#### 8.4.2.8 EAPI void* ecore_dlist_goto_last (Ecore_DList * *list*)

Move the pointer to the current item to the last item.

**Parameters:**

***list,:*** the list to move the current item pointer to the last

**Returns:**

Returns a pointer to the last item in the list , NULL if empty.

#### 8.4.2.9 EAPI int ecore_dlist_index (Ecore_DList * *list*) [inline]

Retrieves the index of the current node of the given doubly linked list.

**Parameters:**

***list*** The given doubly linked list.

**Returns:**

The index of the current node.

#### 8.4.2.10 EAPI int ecore_dlist_is_empty (Ecore_DList * *list*)

Returns whether there is anything in the given doubly linked list.

**Parameters:**

***list*** The given doubly linked list.

**Returns:**

`TRUE` if there are nodes, `FALSE` otherwise.

#### 8.4.2.11 EAPI int ecore_dlist_mergesort (Ecore_DList * *list*, Ecore_Compare_Cb *compare*, char *order*)

Sort data in `list` using the compare function `compare`.

**Parameters:**

***list*** The list.
***compare*** The function to compare the data of `list`
***order*** The sort direction

**Returns:**

true on success

Mergesort is a stable, in-place sorting algorithm

#### 8.4.2.12 EAPI void* ecore_dlist_next (Ecore_DList * *list*)

Move to the next item in the list and return current item.

**Parameters:**

***list,:*** the list to move to the next item in.

**Returns:**

Returns data in the current list node, or NULL on error

#### 8.4.2.13 EAPI void* ecore_dlist_previous (Ecore_DList * *list*)

Move to the previous item and return current item.

**Parameters:**

***list,:*** the list to move to the previous item in.

**Returns:**

Returns data in the current list node, or NULL on error

#### 8.4.2.14 EAPI int ecore_dlist_sort (Ecore_List * *list*, Ecore_Compare_Cb *compare*, char *order*)

Sort data in `list` using the compare function `compare`.

**Parameters:**

***list*** The list.

***compare*** The function to compare the data of `list`

***order*** The sort direction

**Returns:**

true on success

This is a wrapper function for mergesort and heapsort. It tries to choose the fastest algorithm depending on the number of notes. Note: The sort may be unstable.

#### 8.4.2.15 EAPI void ecore_hash_dump_graph (Ecore_Hash * *hash*)

Prints the distribution of the given hash table for graphing.

**Parameters:**

***hash*** The given hash table.

#### 8.4.2.16 EAPI void ecore_hash_dump_stats (Ecore_Hash * *hash*)

Prints the distribution of the given hash table for graphing.

**Parameters:**

***hash*** The given hash table.

#### 8.4.2.17 EAPI int ecore_list_clear (Ecore_List * *list*)

Remove all nodes from `list`.

**Parameters:**

***list*** The list.

**Returns:**

Returns `TRUE` on success, `FALSE` on error.

**Note:**

The data for each item on the list is not freed by `ecore_list_clear()`.

#### 8.4.2.18 EAPI void* ecore_list_current (Ecore_List * *list*) [inline]

Retrieve the data pointed to by the current item in `list`.

**Parameters:**

***list*** The list.

**Returns:**

Returns the data at current position, can be `NULL`.

#### 8.4.2.19 EAPI void* ecore_list_find (Ecore_List * *list*, Ecore_Compare_Cb *function*, const void * *user_data*)

Find data in `list` using the compare function `func`.

**Parameters:**

***list*** The list.

***function*** The function to test each node of `list` with

***user_data*** Data to match against (used by `function`)

**Returns:**

the first matching data node, or NULL if none match

#### 8.4.2.20 EAPI void* ecore_list_first (Ecore_List * *list*) [inline]

Retrieve the data pointed to by the first item in `list`.

**Parameters:**

***list*** The list.

**Returns:**

Returns the data at current position, can be `NULL`.

#### 8.4.2.21 EAPI int ecore_list_heapsort (Ecore_List * *list*, Ecore_Compare_Cb *compare*, char *order*)

Sort data in `list` using the compare function `compare`.

**Parameters:**

***list*** The list.

***compare*** The function to compare the data of `list`

***order*** The sort direction

**Returns:**

true on success

Heapsort is a unstable sorting algorithm, it needs to allocate extra memomry, but there for it is for a great number of nodes faster than mergesort

#### 8.4.2.22 EAPI int ecore_list_index (Ecore_List * *list*)

Returns the number of the current node.

**Parameters:**

***list*** The list to return the number of the current node.

**Returns:**

The number of the current node in the list.

#### 8.4.2.23 EAPI int ecore_list_is_empty (Ecore_List * *list*)

Checks the list for any nodes.

**Parameters:**

***list*** The list to check for nodes

**Returns:**

`TRUE` if no nodes in list, `FALSE` if the list contains nodes

#### 8.4.2.24 EAPI void* ecore_list_last (Ecore_List * *list*) [inline]

Retrieve the data pointed to by the last item in `list`.

**Parameters:**

***list*** The list.

**Returns:**

Returns the data at current position, can be `NULL`.

#### 8.4.2.25 EAPI int ecore_list_mergesort (Ecore_List * *list*, Ecore_Compare_Cb *compare*, char *order*)

Sort data in `list` using the compare function `compare`.

**Parameters:**

***list*** The list.
***compare*** The function to compare the data of `list`
***order*** The sort direction

**Returns:**

true on success

Mergesort is a stable, in-place sorting algorithm

#### 8.4.2.26 EAPI void* ecore_list_next (Ecore_List * *list*) [inline]

Retrieve the data pointed to by the current item, and make the next item the current item.

**Parameters:**

***list*** The list to retrieve data from.

**Returns:**

The current item in the list on success, `NULL` on failure.

#### 8.4.2.27 EAPI int ecore_list_nodes (Ecore_List * *list*)

Find the number of nodes in the list.

**Parameters:**

***list*** The list to find the number of nodes

**Returns:**

The number of nodes in the list.

#### 8.4.2.28 EAPI int ecore_list_set_free_cb (Ecore_List * *list*, Ecore_Free_Cb *free_func*)

Set the function for freeing data.

**Parameters:**

***list*** The list that will use this function when nodes are destroyed.

***free_func*** The function that will free the key data.

**Returns:**

`TRUE` on successful set, `FALSE` otherwise.

#### 8.4.2.29 EAPI int ecore_list_sort (Ecore_List * *list*, Ecore_Compare_Cb *compare*, char *order*)

Sort data in `list` using the compare function `compare`.

**Parameters:**

***list*** The list.

***compare*** The function to compare the data of `list`

***order*** The sort direction

**Returns:**

true on success

This is a wrapper function for mergesort and heapsort. It tries to choose the fastest algorithm depending on the number of notes. Note: The sort may be unstable.

#### 8.4.2.30 EAPI int ecore_sheap_change (Ecore_Sheap * *heap*, void * *item*, void * *newval*)

Change the value of the specified item in the heap.

**Parameters:**

***heap*** The heap to search for the item to change

***item*** The item in the heap to change

***newval*** The new value assigned to the item in the heap

**Returns:**

TRUE on success, FALSE on failure.

**Note:**

The heap does not free the old data since it must be passed in, so the caller can perform the free if desired.

#### 8.4.2.31 EAPI void ecore_sheap_destroy (Ecore_Sheap * *heap*)

Free up the memory used by the heap.

Frees the memory used by *heap*, calls the destroy function on each data item if necessary.

**Parameters:**

***heap*** The heap to be freed

#### 8.4.2.32 EAPI void* ecore_sheap_extract (Ecore_Sheap * *heap*)

Extract the item at the top of the heap.

**Parameters:**

***heap*** The heap to remove the top item

**Returns:**

The top item of the heap on success, NULL on failure.

**Note:**

The extract function maintains the heap properties after the extract.

#### 8.4.2.33 EAPI void* ecore_sheap_extreme (Ecore_Sheap * *heap*)

Examine the item at the top of the heap.

**Parameters:**

***heap*** The heap to examine the top item

**Returns:**

The top item of the heap on success, NULL on failure.

**Note:**

The function does not alter the heap.

#### 8.4.2.34 EAPI int ecore_sheap_init (Ecore_Sheap * *heap*, Ecore_Compare_Cb *compare*, int *size*)

Initialize a binary heap to default values.

**Parameters:**

***heap*** The heap to initialize

***compare*** The function for comparing keys, NULL for direct comparison

***size*** The number of elements to allow in the heap

**Returns:**

TRUE on success, FALSE on failure

#### 8.4.2.35 EAPI int ecore_sheap_insert (Ecore_Sheap * *heap*, void * *data*)

Insert new data into the heap.

**Parameters:**

***heap*** The heap to insert *data*.

***data*** The data to add to *heap*.

**Returns:**

TRUE on success, NULL on failure. Increases the size of the heap if it becomes larger than available space.

#### 8.4.2.36 EAPI Ecore_Sheap* ecore_sheap_new (Ecore_Compare_Cb *compare*, int *size*)

Allocate and initialize a new binary heap.

**Parameters:**

***compare*** The function for comparing keys, NULL for direct comparison

***size*** The number of elements to allow in the heap

**Returns:**

A pointer to the newly allocated binary heap on success, NULL on failure.

#### 8.4.2.37 EAPI int ecore_sheap_set_compare (Ecore_Sheap * *heap*, Ecore_Compare_Cb *compare*)

Change the comparison function for the heap.

**Parameters:**

***heap*** The heap to change comparison function

***compare*** The new function for comparing nodes

**Returns:**

TRUE on success, FALSE on failure.

The comparison function is changed to and the heap is heapified by the new comparison.

#### 8.4.2.38 EAPI int ecore_sheap_set_free_cb (Ecore_Sheap * *heap*, Ecore_Free_Cb *free_func*)

Set the function for freeing data.

**Parameters:**

***heap*** The heap that will use this function when nodes are destroyed.

***free_func*** The function that will free the key data.

**Returns:**

`TRUE` on successful set, `FALSE` otherwise.

#### 8.4.2.39 EAPI void ecore_sheap_set_order (Ecore_Sheap * *heap*, char *order*)

Change the order of the heap.

**Parameters:**

***heap*** The heap to change the order

***order*** The new order of the heap

Changes the heap order of and re-heapifies the data to this new order. The default order is a min heap.

#### 8.4.2.40 EAPI void ecore_sheap_sort (Ecore_Sheap * *heap*)

Sort the data in the heap.

**Parameters:**

***heap*** The heap to be sorted

Sorts the data in the heap into the order that is used for the heap's data.

#### 8.4.2.41 EAPI int ecore_str_compare (const void * *key1*, const void * *key2*)

Perform a string comparison of two keys values.

**Parameters:**

***key1*** The first key to compare

***key2*** The second key to compare

**Returns:**

A strcmp style value to indicate the larger key

#### 8.4.2.42 EAPI unsigned int ecore_str_hash (const void * *key*)

Compute the hash value of a string.

**Parameters:**

***key*** A pointer to the string to compute a hash value

**Returns:**

A computed hash value for *key*.

#### 8.4.2.43 EAPI void ecore_strbuf_append (Ecore_Strbuf * *buf*, const char * *str*)

Append a string to a buffer, reallocating as necessary.

**Parameters:**

***buf*** the Ecore_Strbuf to append to

***str*** the string to append

#### 8.4.2.44 EAPI void ecore_strbuf_append_char (Ecore_Strbuf * *buf*, char *c*)

Append a character to a string buffer, reallocating as necessary.

**Parameters:**

***buf*** the Ecore_Strbuf to append to

***c*** the char to append

#### 8.4.2.45 EAPI void ecore_strbuf_free (Ecore_Strbuf * *buf*)

Free a string buffer.

**Parameters:**

***buf*** the buffer to free

#### 8.4.2.46 EAPI void ecore_strbuf_insert (Ecore_Strbuf * *buf*, const char * *str*, size_t *pos*)

Insert a string to a buffer, reallocating as necessary.

**Parameters:**

***buf*** the Ecore_Strbuf to insert

***str*** the string to insert

***pos*** the position to insert the string

#### 8.4.2.47 EAPI size_t ecore_strbuf_length_get (Ecore_Strbuf * *buf*)

Retrieve the length of the string buffer content.

**Parameters:**

***buf*** the buffer

#### 8.4.2.48 EAPI int ecore_strbuf_replace (Ecore_Strbuf * *buf*, const char * *str*, const char * *with*, unsigned int *n*)

Replace the n-th string with an other string.

**Parameters:**

***buf*** the Ecore_Strbuf to work with

***str*** the string to replace

***with*** the replaceing string

***n*** the number of the fitting string

**Returns:**

true on success

#### 8.4.2.49 EAPI int ecore_strbuf_replace_all (Ecore_Strbuf * *buf*, const char * *str*, const char * *with*)

Replace all strings with an other string.

**Parameters:**

***buf*** the Ecore_Strbuf to work with

***str*** the string to replace

***with*** the replaceing string

**Returns:**

how often the string was replaced

#### 8.4.2.50 EAPI const char* ecore_strbuf_string_get (Ecore_Strbuf * *buf*)

Retrieve a pointer to the contents of a string buffer.

**Parameters:**

***buf*** the buffer

This pointer must not be modified, and will no longer be valid if the Ecore_Strbuf is modified.

#### 8.4.2.51 EAPI int ecore_string_init (void)

Initialize the ecore string internal structure.

**Returns:**

Zero on failure, non-zero on successful initialization.

#### 8.4.2.52 EAPI int ecore_tree_add_node (Ecore_Tree * *tree*, Ecore_Tree_Node * *node*)

Place a node in the tree.

**Parameters:**

***tree*** The tree to add *node*.

***node*** The node to add to *tree*.

**Returns:**

TRUE on a successful add, FALSE otherwise.

#### 8.4.2.53 EAPI int ecore_tree_destroy (Ecore_Tree * *tree*)

Free the tree and it's stored data.

**Parameters:**

***tree,:*** the tree to destroy

**Returns:**

Returns TRUE if tree destroyed successfully, FALSE if not.

#### 8.4.2.54 EAPI int ecore_tree_for_each_node (Ecore_Tree * *tree*, Ecore_For_Each *for_each_func*, void * *user_data*)

Execute the function for each node in the tree.

**Parameters:**

***tree,:*** the tree to traverse

***for_each_func,:*** the function to execute for each node

***user_data,:*** data passed to each for_each_func call

**Returns:**

Returns TRUE on success, FALSE on failure.

#### 8.4.2.55 EAPI int ecore_tree_for_each_node_value (Ecore_Tree * *tree*, Ecore_For_Each *for_each_func*, void * *user_data*)

Execute function for each value in the tree.

**Parameters:**

***tree,:*** the tree to traverse

***for_each_func,:*** the function to execute for each value in the tree

***user_data,:*** data passed to each for_each_func call

**Returns:**

Returns TRUE on success, FALSE on failure.

#### 8.4.2.56 EAPI void* ecore_tree_get (Ecore_Tree * *tree*, void * *key*)

Return the value corresponding to key.

**Parameters:**

***tree,:*** the tree to search

***key,:*** the key to search for in *tree*

**Returns:**

Returns the value corresponding to the key if found, otherwise NULL.

#### 8.4.2.57 EAPI void* ecore_tree_get_closest_larger (Ecore_Tree * *tree*, void * *key*)

Find the closest value greater than or equal to the key.

**Parameters:**

***tree*** The tree to search.

***key*** The key to search for in *tree*.

**Returns:**

NULL if no valid nodes, otherwise the node greater than or equal to the key

#### 8.4.2.58 EAPI void* ecore_tree_get_closest_smaller (Ecore_Tree * *tree*, void * *key*)

Find the closest value <= key.

**Parameters:**

***tree*** the tree to search

***key*** the key to search for in tree

**Returns:**

Returns NULL if no valid nodes, otherwise the node <= key

#### 8.4.2.59 EAPI Ecore_Tree_Node* ecore_tree_get_node (Ecore_Tree * *tree*, void * *key*)

Return the node corresponding to key.

**Parameters:**

***tree,:*** the tree to search

***key,:*** the key to search for in the tree

**Returns:**

Returns the node corresponding to the key if found, otherwise NULL.

#### 8.4.2.60 EAPI int ecore_tree_init (Ecore_Tree * *new_tree*, Ecore_Compare_Cb *compare_func*)

Initialize a tree structure to some sane initial values.

**Parameters:**

***new_tree,:*** the new tree structure to be initialized

***compare_func,:*** the function used to compare node keys

**Returns:**

Returns TRUE on successful initialization, FALSE on an error

#### 8.4.2.61 EAPI int ecore_tree_is_empty (Ecore_Tree * *tree*)

Test to see if the tree has any nodes.

**Parameters:**

***tree,:*** the tree to check for nodes

**Returns:**

Returns TRUE if no nodes exist, FALSE otherwise

#### 8.4.2.62 EAPI Ecore_Tree* ecore_tree_new (Ecore_Compare_Cb *compare_func*)

Allocate a new tree structure.

**Parameters:**

***compare_func,:*** function used to compare the two values

**Returns:**

Returns NULL if the operation fails, otherwise the new tree

#### 8.4.2.63 EAPI int ecore_tree_remove (Ecore_Tree * *tree*, void * *key*)

Remove the key from the tree.

**Parameters:**

***tree*** The tree to remove *key*.

***key*** The key to remove from *tree*.

**Returns:**

TRUE on a successful remove, FALSE otherwise.

#### 8.4.2.64 EAPI int ecore_tree_remove_node (Ecore_Tree * *tree*, Ecore_Tree_Node * *node*)

Remove the node from the tree.

**Parameters:**

***tree*** The tree to remove *node* from.

***node*** The node to remove from *tree*.

**Returns:**

TRUE on a successful remove, FALSE otherwise.

#### 8.4.2.65 EAPI int ecore_tree_set (Ecore_Tree * *tree*, void * *key*, void * *value*)

Set the value associated with key to *value*.

**Parameters:**

***tree*** The tree that contains the key/value pair.

***key*** The key to identify which node to set a value.

***value*** Value to set the found node.

**Returns:**

TRUE if successful, FALSE if not.

## 8.5 Ecore_Desktop.h File Reference

The file that provides the freedesktop.org desktop, icon, and menu functions.

### Functions

- char * ecore_desktop_paths_file_find (Ecore_List *paths, const char *file, int sub, int(*func)(void *data, const char *path), void *data)

  *Search for a file in fdo compatible locations.*

- Ecore_Hash * ecore_desktop_paths_to_hash (const char *paths)

  *Split a list of paths into an Ecore_Hash.*

- Ecore_List * ecore_desktop_paths_to_list (const char *paths)

  *Split a list of paths into an Ecore_Hash.*

- EAPI int ecore_desktop_init (void)

  *Setup what ever needs to be setup to support Ecore_Desktop.*

- EAPI int ecore_desktop_shutdown (void)

  *Tear down what ever needs to be torn down to support Ecore_Desktop.*

- Ecore_Hash * ecore_desktop_ini_get (const char *file)

  *Get the contents of a .ini style file.*

- Ecore_Desktop * ecore_desktop_get (const char *file, const char *lang)

  *Get the contents of a .desktop file.*

- void ecore_desktop_destroy (Ecore_Desktop *desktop)

  *Free whatever resources are used by an Ecore_Desktop.*

- EAPI int ecore_desktop_icon_init (void)

  *Setup what ever needs to be setup to support ecore_desktop_icon.*

- EAPI int ecore_desktop_icon_shutdown (void)

  *Tear down what ever needs to be torn down to support ecore_desktop_ycon.*

- EAPI char * ecore_desktop_icon_find (const char *icon, const char *icon_size, const char *icon_theme)

  *Find the path to an icon.*

- void ecore_desktop_icon_theme_destroy (Ecore_Desktop_Icon_Theme *icon_theme)

  *Free whatever resources are used by an Ecore_Desktop_Icon_Theme.*

- Ecore_Desktop_Tree * ecore_desktop_menu_get (char *file)

  *Decode a freedesktop.org menu XML jungle.*

- char * ecore_desktop_home_get (void)

  *Get and massage the users home directory.*

### 8.5.1 Detailed Description

The file that provides the freedesktop.org desktop, icon, and menu functions.

This header provides the Ecore_Desktop freedesktop.org desktop, icon, and menu handling functions, as well as ancillary functions for searching freedesktop.org specific paths. Other freedesktop.org specifications make use of similar files, paths, and icons, implementors can use / extend this code to suit.

Ecore_Desktop is not for every freedesktop.org specification, just those that are associated with .desktop files.

For path searching details, see Ecore_Desktop_Paths_Group.

For desktop file details, see .desktop file Functions.

For icon theme details, see icon theme Functions.

For menu file details, see menu Functions.

### 8.5.2 Function Documentation

#### 8.5.2.1 char* ecore_desktop_paths_file_find (Ecore_List * *paths*, const char * *file*, int *sub*, int(*)(void *data, const char *path) *func*, void * *data*)

Search for a file in fdo compatible locations.

This will search through all the diretories of a particular type, looking for the file. It will recurse into subdirectories. If func is NULL, then only the first file found will be returned. If func is defined, then each file found will be passed to func, until func returns 1.

The returned string will have to be freed eventually.

**Parameters:**

***type*** The type of directories to search.

***file*** The file to search for.

***sub*** Levels of sub directories to search, -1 = all, 0 = none.

***func*** A function to call for each file found.

***data*** A pointer to pass on to func.

#### 8.5.2.2 Ecore_Hash* ecore_desktop_paths_to_hash (const char * *paths*)

Split a list of paths into an Ecore_Hash.

The list of paths can use any one of ;:, to seperate the paths. You can also escape the :;, with \.

**Parameters:**

***paths*** A list of paths.

#### 8.5.2.3 Ecore_List* ecore_desktop_paths_to_list (const char * *paths*)

Split a list of paths into an Ecore_Hash.

The list of paths can use any one of ;:, to seperate the paths. You can also escape the :;, with \.

**Parameters:**

***paths*** A list of paths.

## 8.6 Ecore_Evas.h File Reference

Evas wrapper functions.

### Functions

- EAPI int ecore_evas_engine_type_supported_get (Ecore_Evas_Engine_Type engine)

  *Query if a particular renginering engine target has support.*

- EAPI int ecore_evas_init (void)

  *Init the Evas system.*

- EAPI int ecore_evas_shutdown (void)

  *Shut down the Evas system.*

- EAPI Ecore_Evas * ecore_evas_software_x11_new (const char *disp_name, Ecore_X_Window parent, int x, int y, int w, int h)

  *To be documented.*

- EAPI Ecore_X_Window ecore_evas_software_x11_window_get (Ecore_Evas *ee)

  *To be documented.*

- EAPI Ecore_X_Window ecore_evas_software_x11_subwindow_get (Ecore_Evas *ee)

  *To be documented.*

- EAPI void ecore_evas_software_x11_direct_resize_set (Ecore_Evas *ee, int on)

  *To be documented.*

- EAPI int ecore_evas_software_x11_direct_resize_get (Ecore_Evas *ee)

  *To be documented.*

- EAPI void ecore_evas_software_x11_extra_event_window_add (Ecore_Evas *ee, Ecore_X_Window win)

  *To be documented.*

- EAPI Ecore_Evas * ecore_evas_gl_x11_new (const char *disp_name, Ecore_X_Window parent, int x, int y, int w, int h)

  *To be documented.*

- EAPI Ecore_X_Window ecore_evas_gl_x11_window_get (Ecore_Evas *ee)

  *To be documented.*

- EAPI Ecore_X_Window ecore_evas_gl_x11_subwindow_get (Ecore_Evas *ee)

  *To be documented.*

- EAPI void ecore_evas_gl_x11_direct_resize_set (Ecore_Evas *ee, int on)

*To be documented.*

- EAPI int ecore_evas_gl_x11_direct_resize_get (Ecore_Evas *ee)

  *To be documented.*

- EAPI void ecore_evas_gl_x11_extra_event_window_add (Ecore_Evas *ee, Ecore_X_Window win)

  *To be documented.*

- EAPI Ecore_Evas * ecore_evas_xrender_x11_new (const char *disp_name, Ecore_X_Window parent, int x, int y, int w, int h)

  *To be documented.*

- EAPI Ecore_X_Window ecore_evas_xrender_x11_window_get (Ecore_Evas *ee)

  *To be documented.*

- EAPI Ecore_X_Window ecore_evas_xrender_x11_subwindow_get (Ecore_Evas *ee)

  *To be documented.*

- EAPI void ecore_evas_xrender_x11_direct_resize_set (Ecore_Evas *ee, int on)

  *To be documented.*

- EAPI int ecore_evas_xrender_x11_direct_resize_get (Ecore_Evas *ee)

  *To be documented.*

- EAPI void ecore_evas_xrender_x11_extra_event_window_add (Ecore_Evas *ee, Ecore_X_Window win)

  *To be documented.*

- EAPI Ecore_Evas * ecore_evas_fb_new (char *disp_name, int rotation, int w, int h)

  *To be documented.*

- EAPI Ecore_Evas * ecore_evas_buffer_new (int w, int h)

  *To be documented.*

- EAPI Ecore_Evas * ecore_evas_ecore_evas_get (Evas *e)

  *Return the Ecore_Evase for this Evas.*

- EAPI void ecore_evas_free (Ecore_Evas *ee)

  *Free an Ecore_Evas.*

- EAPI void ecore_evas_callback_resize_set (Ecore_Evas *ee, void(*func)(Ecore_Evas *ee))

  *Set a callback for Ecore_Evas resize events.*

- EAPI void ecore_evas_callback_move_set (Ecore_Evas *ee, void(*func)(Ecore_Evas *ee))

  *Set a callback for Ecore_Evas move events.*

- EAPI void ecore_evas_callback_show_set (Ecore_Evas *ee, void(*func)(Ecore_Evas *ee))

*Set a callback for Ecore_Evas show events.*

- EAPI void ecore_evas_callback_hide_set (Ecore_Evas *ee, void(*func)(Ecore_Evas *ee))

  *Set a callback for Ecore_Evas hide events.*

- EAPI void ecore_evas_callback_delete_request_set (Ecore_Evas *ee, void(*func)(Ecore_Evas *ee))

  *Set a callback for Ecore_Evas delete request events.*

- EAPI void ecore_evas_callback_destroy_set (Ecore_Evas *ee, void(*func)(Ecore_Evas *ee))

  *Set a callback for Ecore_Evas destroy events.*

- EAPI void ecore_evas_callback_focus_in_set (Ecore_Evas *ee, void(*func)(Ecore_Evas *ee))

  *Set a callback for Ecore_Evas focus in events.*

- EAPI void ecore_evas_callback_focus_out_set (Ecore_Evas *ee, void(*func)(Ecore_Evas *ee))

  *Set a callback for Ecore_Evas focus out events.*

- EAPI void ecore_evas_callback_sticky_set (Ecore_Evas *ee, void(*func)(Ecore_Evas *ee))

  *Set a callback for Ecore_Evas sticky events.*

- EAPI void ecore_evas_callback_unsticky_set (Ecore_Evas *ee, void(*func)(Ecore_Evas *ee))

  *Set a callback for Ecore_Evas un-sticky events.*

- EAPI void ecore_evas_callback_mouse_in_set (Ecore_Evas *ee, void(*func)(Ecore_Evas *ee))

  *Set a callback for Ecore_Evas mouse in events.*

- EAPI void ecore_evas_callback_mouse_out_set (Ecore_Evas *ee, void(*func)(Ecore_Evas *ee))

  *Set a callback for Ecore_Evas mouse out events.*

- EAPI void ecore_evas_callback_pre_render_set (Ecore_Evas *ee, void(*func)(Ecore_Evas *ee))

  *Set a callback for Ecore_Evas mouse pre render events.*

- EAPI void ecore_evas_callback_post_render_set (Ecore_Evas *ee, void(*func)(Ecore_Evas *ee))

  *Set a callback for Ecore_Evas mouse post render events.*

- EAPI Evas * ecore_evas_get (Ecore_Evas *ee)

  *Get an Ecore_Evas's Evas.*

- EAPI void ecore_evas_move (Ecore_Evas *ee, int x, int y)

  *Move an Ecore_Evas.*

- EAPI void ecore_evas_managed_move (Ecore_Evas *ee, int x, int y)
  *Provide Managed move co-ordinates for an Ecore_Evas.*
- EAPI void ecore_evas_resize (Ecore_Evas *ee, int w, int h)
  *Resize an Ecore_Evas.*
- EAPI void ecore_evas_move_resize (Ecore_Evas *ee, int x, int y, int w, int h)
  *Resize an Ecore_Evas.*
- EAPI void ecore_evas_geometry_get (Ecore_Evas *ee, int *x, int *y, int *w, int *h)
  *Get the geometry of an Ecore_Evas.*
- EAPI void ecore_evas_rotation_set (Ecore_Evas *ee, int rot)
  *Set the rotation of an Ecore_Evas' window.*
- EAPI int ecore_evas_rotation_get (Ecore_Evas *ee)
  *Set the rotation of an Ecore_Evas' window.*
- EAPI void ecore_evas_shaped_set (Ecore_Evas *ee, int shaped)
  *Set whether an Ecore_Evas is shaped or not.*
- EAPI int ecore_evas_shaped_get (Ecore_Evas *ee)
  *Query whether an Ecore_Evas is shaped or not.*
- EAPI void ecore_evas_alpha_set (Ecore_Evas *ee, int alpha)
  *Set whether an Ecore_Evas has an alpha channel or not.*
- EAPI int ecore_evas_alpha_get (Ecore_Evas *ee)
  *Query whether an Ecore_Evas is alpha or not.*
- EAPI void ecore_evas_show (Ecore_Evas *ee)
  *Show an Ecore_Evas' window.*
- EAPI void ecore_evas_hide (Ecore_Evas *ee)
  *Hide an Ecore_Evas' window.*
- EAPI int ecore_evas_visibility_get (Ecore_Evas *ee)
  *Query whether an Ecore_Evas' window is visible or not.*
- EAPI void ecore_evas_raise (Ecore_Evas *ee)
  *Raise and Ecore_Evas' window.*
- EAPI void ecore_evas_lower (Ecore_Evas *ee)
  *Lower an Ecore_Evas' window.*
- EAPI void ecore_evas_title_set (Ecore_Evas *ee, const char *t)
  *Set the title of an Ecore_Evas' window.*
- EAPI const char * ecore_evas_title_get (Ecore_Evas *ee)

*Get the title of an Ecore_Evas' window.*

- EAPI void ecore_evas_name_class_set (Ecore_Evas *ee, const char *n, const char *c)

  *Set the name and class of an Ecore_Evas' window.*

- EAPI void ecore_evas_name_class_get (Ecore_Evas *ee, const char **n, const char **c)

  *Get the name and class of an Ecore_Evas' window* `ee` *The Ecore_Evas to query* `n` *A pointer to a string to place the name in.*

- EAPI void ecore_evas_size_min_set (Ecore_Evas *ee, int w, int h)

  *Set the min size of an Ecore_Evas' window.*

- EAPI void ecore_evas_size_min_get (Ecore_Evas *ee, int *w, int *h)

  *Get the min size of an Ecore_Evas' window.*

- EAPI void ecore_evas_size_max_set (Ecore_Evas *ee, int w, int h)

  *Set the max size of an Ecore_Evas' window.*

- EAPI void ecore_evas_size_max_get (Ecore_Evas *ee, int *w, int *h)

  *Get the max size of an Ecore_Evas' window.*

- EAPI void ecore_evas_size_base_set (Ecore_Evas *ee, int w, int h)

  *Set the base size of an Ecore_Evas' window.*

- EAPI void ecore_evas_size_base_get (Ecore_Evas *ee, int *w, int *h)

  *Get the base size of an Ecore_Evas' window.*

- EAPI void ecore_evas_size_step_set (Ecore_Evas *ee, int w, int h)

  *Set the step size of an Ecore_Evas.*

- EAPI void ecore_evas_size_step_get (Ecore_Evas *ee, int *w, int *h)

  *Get the step size of an Ecore_Evas' window.*

- EAPI void ecore_evas_cursor_set (Ecore_Evas *ee, const char *file, int layer, int hot_x, int hot_y)

  *Set the cursor of an Ecore_Evas.*

- EAPI void ecore_evas_cursor_get (Ecore_Evas *ee, char **file, int *layer, int *hot_x, int *hot_y)

  *Get information about an Ecore_Evas' cursor.*

- EAPI void ecore_evas_layer_set (Ecore_Evas *ee, int layer)

  *Set the layer of an Ecore_Evas' window.*

- EAPI int ecore_evas_layer_get (Ecore_Evas *ee)

  *Get the layer of an Ecore_Evas' window.*

- EAPI void ecore_evas_focus_set (Ecore_Evas *ee, int on)

  *Set the focus of an Ecore_Evas' window.*

- EAPI int ecore_evas_focus_get (Ecore_Evas *ee)
  *Query whether an Ecore_Evas' window is focused or not.*
- EAPI void ecore_evas_iconified_set (Ecore_Evas *ee, int on)
  *Iconify or uniconify an Ecore_Evas' window.*
- EAPI int ecore_evas_iconified_get (Ecore_Evas *ee)
  *Query whether an Ecore_Evas' window is iconified or not.*
- EAPI void ecore_evas_borderless_set (Ecore_Evas *ee, int on)
  *Set whether an Ecore_Evas' window is borderless or not.*
- EAPI int ecore_evas_borderless_get (Ecore_Evas *ee)
  *Query whether an Ecore_Evas' window is borderless or not.*
- EAPI void ecore_evas_override_set (Ecore_Evas *ee, int on)
  *Tell the WM whether or not to ignore an Ecore_Evas' window.*
- EAPI int ecore_evas_override_get (Ecore_Evas *ee)
  *Query whether an Ecore_Evas' window is overridden or not.*
- EAPI void ecore_evas_maximized_set (Ecore_Evas *ee, int on)
  *Maximize (or unmaximize) an Ecore_Evas' window.*
- EAPI int ecore_evas_maximized_get (Ecore_Evas *ee)
  *Query whether an Ecore_Evas' window is maximized or not.*
- EAPI void ecore_evas_fullscreen_set (Ecore_Evas *ee, int on)
  *Set whether or not an Ecore_Evas' window is fullscreen.*
- EAPI int ecore_evas_fullscreen_get (Ecore_Evas *ee)
  *Query whether an Ecore_Evas' window is fullscreen or not.*
- EAPI void ecore_evas_avoid_damage_set (Ecore_Evas *ee, int on)
  *Set whether or not an Ecore_Evas' window should avoid damage.*
- EAPI int ecore_evas_avoid_damage_get (Ecore_Evas *ee)
  *Query whether an Ecore_Evas' window avoids damage or not.*
- EAPI void ecore_evas_withdrawn_set (Ecore_Evas *ee, int withdrawn)
  *Set the withdrawn state of an Ecore_Evas' window.*
- EAPI int ecore_evas_withdrawn_get (Ecore_Evas *ee)
  *Returns the withdrawn state of an Ecore_Evas' window.*
- EAPI void ecore_evas_sticky_set (Ecore_Evas *ee, int sticky)
  *Set the sticky state of an Ecore_Evas window.*
- EAPI int ecore_evas_sticky_get (Ecore_Evas *ee)
  *Returns the sticky state of an Ecore_Evas' window.*

- EAPI void ecore_evas_ignore_events_set (Ecore_Evas *ee, int ignore)
  *Set if this evas should ignore events.*
- EAPI int ecore_evas_ignore_events_get (Ecore_Evas *ee)
  *Returns the ignore state of an Ecore_Evas' window.*

## 8.6.1 Detailed Description

Evas wrapper functions.

## 8.6.2 Function Documentation

### 8.6.2.1 EAPI int ecore_evas_alpha_get (Ecore_Evas * *ee*)

Query whether an Ecore_Evas is alpha or not.

**Parameters:**

***ee*** The Ecore_Evas to query.

**Returns:**

1 if alpha, 0 if not.

This function returns 1 if `ee` is alpha, and 0 if not.

### 8.6.2.2 EAPI void ecore_evas_alpha_set (Ecore_Evas * *ee*, int *alpha*)

Set whether an Ecore_Evas has an alpha channel or not.

**Parameters:**

***ee*** The Ecore_Evas to shape

***shaped*** 1 to add alpha, 0 to not

This function allows one to make an Ecore_Evas translucent using alpha channels. See ecore_evas_shaped_set() for details. The differency with alpha is it supports multiple levels of transparencye, not just a single outline.

### 8.6.2.3 EAPI int ecore_evas_avoid_damage_get (Ecore_Evas * *ee*)

Query whether an Ecore_Evas' window avoids damage or not.

**Parameters:**

***ee*** The Ecore_Evas to set

**Returns:**

1 if ee avoids damage, 0 if not.

#### 8.6.2.4 EAPI void ecore_evas_avoid_damage_set (Ecore_Evas * *ee*, int *on*)

Set whether or not an Ecore_Evas' window should avoid damage.

**Parameters:**

***ee*** The Ecore_Evas

***on*** 1 to avoid damage, 0 to not

This function causes ee to be drawn to a pixmap to avoid recalculations. On expose events it will copy from the pixmap to the window.

#### 8.6.2.5 EAPI int ecore_evas_borderless_get (Ecore_Evas * *ee*)

Query whether an Ecore_Evas' window is borderless or not.

**Parameters:**

***ee*** The Ecore_Evas to set

**Returns:**

1 if ee is borderless, 0 if not.

#### 8.6.2.6 EAPI void ecore_evas_borderless_set (Ecore_Evas * *ee*, int *on*)

Set whether an Ecore_Evas' window is borderless or not.

**Parameters:**

***ee*** The Ecore_Evas

***on*** 1 for borderless, 0 for bordered.

This function makes ee borderless if on is 1, or bordered if on is 0.

#### 8.6.2.7 EAPI Ecore_Evas* ecore_evas_buffer_new (int *w*, int *h*)

To be documented.

FIXME: To be fixed.

#### 8.6.2.8 EAPI void ecore_evas_callback_delete_request_set (Ecore_Evas * *ee*, void(*)(Ecore_Evas *ee) *func*)

Set a callback for Ecore_Evas delete request events.

**Parameters:**

***ee*** The Ecore_Evas to set callbacks on

***func*** The function to call

A call to this function will set a callback on an Ecore_Evas, causing `func` to be called whenever `ee` gets a delete request.

#### 8.6.2.9 EAPI void ecore_evas_callback_destroy_set (Ecore_Evas * *ee*, void(*)(Ecore_Evas *ee) *func*)

Set a callback for Ecore_Evas destroy events.

**Parameters:**

***ee*** The Ecore_Evas to set callbacks on

***func*** The function to call

A call to this function will set a callback on an Ecore_Evas, causing `func` to be called whenever `ee` is destroyed.

#### 8.6.2.10 EAPI void ecore_evas_callback_focus_in_set (Ecore_Evas * *ee*, void(*)(Ecore_Evas *ee) *func*)

Set a callback for Ecore_Evas focus in events.

**Parameters:**

***ee*** The Ecore_Evas to set callbacks on

***func*** The function to call

A call to this function will set a callback on an Ecore_Evas, causing `func` to be called whenever `ee` gets focus.

#### 8.6.2.11 EAPI void ecore_evas_callback_focus_out_set (Ecore_Evas * *ee*, void(*)(Ecore_Evas *ee) *func*)

Set a callback for Ecore_Evas focus out events.

**Parameters:**

***ee*** The Ecore_Evas to set callbacks on

***func*** The function to call

A call to this function will set a callback on an Ecore_Evas, causing `func` to be called whenever `ee` loses focus.

#### 8.6.2.12 EAPI void ecore_evas_callback_hide_set (Ecore_Evas * *ee*, void(*)(Ecore_Evas *ee) *func*)

Set a callback for Ecore_Evas hide events.

**Parameters:**

***ee*** The Ecore_Evas to set callbacks on

***func*** The function to call

A call to this function will set a callback on an Ecore_Evas, causing `func` to be called whenever `ee` is hidden.

#### 8.6.2.13 EAPI void ecore_evas_callback_mouse_in_set (Ecore_Evas * *ee*, void(*)(Ecore_Evas *ee) *func*)

Set a callback for Ecore_Evas mouse in events.

**Parameters:**

***ee*** The Ecore_Evas to set callbacks on

***func*** The function to call

A call to this function will set a callback on an Ecore_Evas, causing `func` to be called whenever the mouse enters `ee`.

#### 8.6.2.14 EAPI void ecore_evas_callback_mouse_out_set (Ecore_Evas * *ee*, void(*)(Ecore_Evas *ee) *func*)

Set a callback for Ecore_Evas mouse out events.

**Parameters:**

***ee*** The Ecore_Evas to set callbacks on

***func*** The function to call

A call to this function will set a callback on an Ecore_Evas, causing `func` to be called whenever the mouse leaves `ee`.

#### 8.6.2.15 EAPI void ecore_evas_callback_move_set (Ecore_Evas * *ee*, void(*)(Ecore_Evas *ee) *func*)

Set a callback for Ecore_Evas move events.

**Parameters:**

***ee*** The Ecore_Evas to set callbacks on

***func*** The function to call

A call to this function will set a callback on an Ecore_Evas, causing `func` to be called whenever `ee` is moved.

#### 8.6.2.16 EAPI void ecore_evas_callback_post_render_set (Ecore_Evas * *ee*, void(*)(Ecore_Evas *ee) *func*)

Set a callback for Ecore_Evas mouse post render events.

**Parameters:**

***ee*** The Ecore_Evas to set callbacks on

***func*** The function to call

A call to this function will set a callback on an Ecore_Evas, causing `func` to be called just after the evas in `ee` is rendered.

#### 8.6.2.17 EAPI void ecore_evas_callback_pre_render_set (Ecore_Evas * *ee*, void(*)(Ecore_Evas *ee) *func*)

Set a callback for Ecore_Evas mouse pre render events.

**Parameters:**

***ee*** The Ecore_Evas to set callbacks on

***func*** The function to call

A call to this function will set a callback on an Ecore_Evas, causing `func` to be called just before the evas in `ee` is rendered.

#### 8.6.2.18 EAPI void ecore_evas_callback_resize_set (Ecore_Evas * *ee*, void(*)(Ecore_Evas *ee) *func*)

Set a callback for Ecore_Evas resize events.

**Parameters:**

***ee*** The Ecore_Evas to set callbacks on

***func*** The function to call

A call to this function will set a callback on an Ecore_Evas, causing `func` to be called whenever `ee` is resized.

#### 8.6.2.19 EAPI void ecore_evas_callback_show_set (Ecore_Evas * *ee*, void(*)(Ecore_Evas *ee) *func*)

Set a callback for Ecore_Evas show events.

**Parameters:**

***ee*** The Ecore_Evas to set callbacks on

***func*** The function to call

A call to this function will set a callback on an Ecore_Evas, causing `func` to be called whenever `ee` is shown.

#### 8.6.2.20 EAPI void ecore_evas_callback_sticky_set (Ecore_Evas * *ee*, void(*)(Ecore_Evas *ee) *func*)

Set a callback for Ecore_Evas sticky events.

**Parameters:**

***ee*** The Ecore_Evas to set callbacks on

***func*** The function to call

A call to this function will set a callback on an Ecore_Evas, causing `func` to be called whenever `ee` becomes sticky.

#### 8.6.2.21 EAPI void ecore_evas_callback_unsticky_set (Ecore_Evas * *ee*, void(*)(Ecore_Evas *ee) *func*)

Set a callback for Ecore_Evas un-sticky events.

**Parameters:**

***ee*** The Ecore_Evas to set callbacks on

***func*** The function to call

A call to this function will set a callback on an Ecore_Evas, causing `func` to be called whenever `ee` becomes un-sticky.

#### 8.6.2.22 EAPI void ecore_evas_cursor_get (Ecore_Evas * *ee*, char ** *file*, int * *layer*, int * *hot_x*, int * *hot_y*)

Get information about an Ecore_Evas' cursor.

**Parameters:**

***ee*** The Ecore_Evas to set

***file*** A pointer to a string to place the cursor file name in.

***layer*** A pointer to an int to place the cursor's layer in..

***hot_x*** A pointer to an int to place the cursor's hot_x coordinate in.

***hot_y*** A pointer to an int to place the cursor's hot_y coordinate in.

This function queries information about an Ecore_Evas' cursor.

#### 8.6.2.23 EAPI void ecore_evas_cursor_set (Ecore_Evas * *ee*, const char * *file*, int *layer*, int *hot_x*, int *hot_y*)

Set the cursor of an Ecore_Evas.

**Parameters:**

***ee*** The Ecore_Evas

***file*** The path to an image file for the cursor

*layer*

***hot_x*** The x coordinate of the cursor's hot spot

***hot_y*** The y coordinate of the cursor's hot spot

This function makes the mouse cursor over `ee` be the image specified by `file`. The actual point within the image that the mouse is at is specified by `hot_x` and `hot_y`, which are coordinates with respect to the top left corner of the cursor image.

#### 8.6.2.24 EAPI Ecore_Evas* ecore_evas_ecore_evas_get (Evas * *e*)

Return the Ecore_Evase for this Evas.

**Parameters:**

***e*** The Evas to get the Ecore_Evas from

**Returns:**

The Ecore_Evas that holds this Evas

#### 8.6.2.25 EAPI int ecore_evas_engine_type_supported_get (Ecore_Evas_Engine_Type *engine*)

Query if a particular renginering engine target has support.

**Parameters:**

***engine*** The engine to check support for

**Returns:**

1 if the particualr engine is supported, 0 if it is not

Query if engine

**Parameters:**

***engine*** is supported by ecore_evas. 1 is returned if it is, and 0 is returned if it is not supported.

#### 8.6.2.26 EAPI Ecore_Evas* ecore_evas_fb_new (char * *disp_name*, int *rotation*, int *w*, int *h*)

To be documented.

FIXME: To be fixed.

#### 8.6.2.27 EAPI int ecore_evas_focus_get (Ecore_Evas * *ee*)

Query whether an Ecore_Evas' window is focused or not.

**Parameters:**

***ee*** The Ecore_Evas to set

**Returns:**

1 if `ee` if focused, 0 if not.

#### 8.6.2.28 EAPI void ecore_evas_focus_set (Ecore_Evas * *ee*, int *on*)

Set the focus of an Ecore_Evas' window.

**Parameters:**

***ee*** The Ecore_Evas

***on*** 1 for focus, 0 to defocus.

This function focuses `ee` if `on` is 1, or defocuses `ee` if `on` is 0.

#### 8.6.2.29 EAPI void ecore_evas_free (Ecore_Evas * *ee*)

Free an Ecore_Evas.

**Parameters:**

***ee*** The Ecore_Evas to free

This frees up any memory used by the Ecore_Evas.

#### 8.6.2.30 EAPI int ecore_evas_fullscreen_get (Ecore_Evas * *ee*)

Query whether an Ecore_Evas' window is fullscreen or not.

**Parameters:**

***ee*** The Ecore_Evas to set

**Returns:**

1 if `ee` is fullscreen, 0 if not.

#### 8.6.2.31 EAPI void ecore_evas_fullscreen_set (Ecore_Evas * *ee*, int *on*)

Set whether or not an Ecore_Evas' window is fullscreen.

**Parameters:**

***ee*** The Ecore_Evas

***on*** 1 fullscreen, 0 not.

This function causes `ee` to be fullscreen if `on` is 1, or not if `on` is 0.

#### 8.6.2.32 EAPI void ecore_evas_geometry_get (Ecore_Evas * *ee*, int * *x*, int * *y*, int * *w*, int * *h*)

Get the geometry of an Ecore_Evas.

**Parameters:**

***ee*** The Ecore_Evas whose geometry y

***x*** A pointer to an int to place the x coordinate in

***y*** A pointer to an int to place the y coordinate in

***w*** A pointer to an int to place the w size in

***h*** A pointer to an int to place the h size in

This function takes four pointers to (already allocated) ints, and places the geometry of `ee` in them.

```
int x, y, w, h;
ecore_evas_geometry_get(ee, &x, &y, &w, &h);
```

#### 8.6.2.33 EAPI Evas* ecore_evas_get (Ecore_Evas * *ee*)

Get an Ecore_Evas's Evas.

**Parameters:**

***ee*** The Ecore_Evas whose Evas you wish to get

**Returns:**

The Evas wrapped by `ee`

This function returns the Evas contained within `ee`.

#### 8.6.2.34 EAPI int ecore_evas_gl_x11_direct_resize_get (Ecore_Evas * *ee*)

To be documented.

FIXME: To be fixed.

#### 8.6.2.35 EAPI void ecore_evas_gl_x11_direct_resize_set (Ecore_Evas * *ee*, int *on*)

To be documented.

FIXME: To be fixed.

#### 8.6.2.36 EAPI void ecore_evas_gl_x11_extra_event_window_add (Ecore_Evas * *ee*, Ecore_X_Window *win*)

To be documented.

FIXME: To be fixed.

#### 8.6.2.37 EAPI Ecore_Evas* ecore_evas_gl_x11_new (const char * *disp_name*, Ecore_X_Window *parent*, int *x*, int *y*, int *w*, int *h*)

To be documented.

FIXME: To be fixed.

#### 8.6.2.38 EAPI Ecore_X_Window ecore_evas_gl_x11_subwindow_get (Ecore_Evas * *ee*)

To be documented.

FIXME: To be fixed.

#### 8.6.2.39 EAPI Ecore_X_Window ecore_evas_gl_x11_window_get (Ecore_Evas * *ee*)

To be documented.

FIXME: To be fixed.

#### 8.6.2.40 EAPI void ecore_evas_hide (Ecore_Evas * *ee*)

Hide an Ecore_Evas' window.

**Parameters:**

*ee* The Ecore_Evas to show.

This function makes ee hidden.

#### 8.6.2.41 EAPI int ecore_evas_iconified_get (Ecore_Evas * *ee*)

Query whether an Ecore_Evas' window is iconified or not.

**Parameters:**

*ee* The Ecore_Evas to set

**Returns:**

1 if ee is iconified, 0 if not.

#### 8.6.2.42 EAPI void ecore_evas_iconified_set (Ecore_Evas * *ee*, int *on*)

Iconify or uniconify an Ecore_Evas' window.

**Parameters:**

***ee*** The Ecore_Evas

***on*** 1 to iconify, 0 to uniconify.

This function iconifies ee if on is 1, or uniconifies ee if on is 0.

#### 8.6.2.43 EAPI int ecore_evas_ignore_events_get (Ecore_Evas * *ee*)

Returns the ignore state of an Ecore_Evas' window.

**Parameters:**

***ee*** The Ecore_Evas whose window's ignore events state is returned.

**Returns:**

The Ecore_Evas window's ignore state.

#### 8.6.2.44 EAPI void ecore_evas_ignore_events_set (Ecore_Evas * *ee*, int *ignore*)

Set if this evas should ignore events.

**Parameters:**

***ee*** The Ecore_Evas whose window's to ignore events

***sticky*** The Ecore_Evas new ignore state

#### 8.6.2.45 EAPI int ecore_evas_init (void)

Init the Evas system.

**Returns:**

greater than 0 on success, 0 on failure

Set up the Evas wrapper system.

#### 8.6.2.46 EAPI int ecore_evas_layer_get (Ecore_Evas * *ee*)

Get the layer of an Ecore_Evas' window.

**Parameters:**

***ee*** The Ecore_Evas to set

**Returns:**

the layer `ee's` window is on.

#### 8.6.2.47 EAPI void ecore_evas_layer_set (Ecore_Evas * *ee*, int *layer*)

Set the layer of an Ecore_Evas' window.

**Parameters:**

***ee*** The Ecore_Evas

***layer*** The layer to put `ee` on.

This function moves `ee` to the layer `layer`.

#### 8.6.2.48 EAPI void ecore_evas_lower (Ecore_Evas * *ee*)

Lower an Ecore_Evas' window.

**Parameters:**

***ee*** The Ecore_Evas to raise.

This functions lowers the Ecore_Evas to the back.

#### 8.6.2.49 EAPI void ecore_evas_managed_move (Ecore_Evas * *ee*, int *x*, int *y*)

Provide Managed move co-ordinates for an Ecore_Evas.

**Parameters:**

***ee*** The Ecore_Evas to move

***x*** The x coordinate to set as the managed location

***y*** The y coordinate to set as the managed location

This sets the managed geometry position of the `ee` to (`x`, `y`)

#### 8.6.2.50 EAPI int ecore_evas_maximized_get (Ecore_Evas * *ee*)

Query whether an Ecore_Evas' window is maximized or not.

**Parameters:**

***ee*** The Ecore_Evas to set

**Returns:**

1 if `ee` is maximized, 0 if not.

#### 8.6.2.51 EAPI void ecore_evas_maximized_set (Ecore_Evas * *ee*, int *on*)

Maximize (or unmaximize) an Ecore_Evas' window.

**Parameters:**

***ee*** The Ecore_Evas

***on*** 1 to maximize, 0 to unmaximize.

This function maximizes `ee` if `on` is 1, or unmaximizes `ee` if `on` is 0.

#### 8.6.2.52 EAPI void ecore_evas_move (Ecore_Evas * *ee*, int *x*, int *y*)

Move an Ecore_Evas.

**Parameters:**

***ee*** The Ecore_Evas to move

***x*** The x coordinate to move to

***y*** The y coordinate to move to

This moves `ee` to the screen coordinates (`x`, `y`)

#### 8.6.2.53 EAPI void ecore_evas_move_resize (Ecore_Evas * *ee*, int *x*, int *y*, int *w*, int *h*)

Resize an Ecore_Evas.

**Parameters:**

***ee*** The Ecore_Evas to move

***x*** The x coordinate to move to

***y*** The y coordinate to move to

***w*** The w coordinate to resize to

***h*** The h coordinate to resize to

This moves `ee` to the screen coordinates (`x`, `y`) and resizes it to `w` x `h`.

#### 8.6.2.54 EAPI void ecore_evas_name_class_get (Ecore_Evas * *ee*, const char ** *n*, const char ** *c*)

Get the name and class of an Ecore_Evas' window `ee` The Ecore_Evas to query `n` A pointer to a string to place the name in.

`c` A pointer to a string to place the class in.

This function gets puts the name of `ee` into `n`, and its class into `c`.

#### 8.6.2.55 EAPI void ecore_evas_name_class_set (Ecore_Evas * *ee*, const char * *n*, const char * *c*)

Set the name and class of an Ecore_Evas' window.

**Parameters:**

***ee*** the Ecore_Evas

***n*** the name

***c*** the class

This function sets the name of `ee` to `n`, and its class to `c`.

#### 8.6.2.56 EAPI int ecore_evas_override_get (Ecore_Evas * *ee*)

Query whether an Ecore_Evas' window is overridden or not.

**Parameters:**

***ee*** The Ecore_Evas to set

**Returns:**

1 if `ee` is overridden, 0 if not.

#### 8.6.2.57 EAPI void ecore_evas_override_set (Ecore_Evas * *ee*, int *on*)

Tell the WM whether or not to ignore an Ecore_Evas' window.

**Parameters:**

***ee*** The Ecore_Evas

***on*** 1 to ignore, 0 to not.

This function causes the window manager to ignore `ee` if `on` is 1, or not ignore `ee` if `on` is 0.

#### 8.6.2.58 EAPI void ecore_evas_raise (Ecore_Evas * *ee*)

Raise and Ecore_Evas' window.

**Parameters:**

*ee* The Ecore_Evas to raise.

This functions raises the Ecore_Evas to the front.

#### 8.6.2.59 EAPI void ecore_evas_resize (Ecore_Evas * *ee*, int *w*, int *h*)

Resize an Ecore_Evas.

**Parameters:**

*ee* The Ecore_Evas to move
*w* The w coordinate to resize to
*h* The h coordinate to resize to

This resizes `ee` to `w` x `h`

#### 8.6.2.60 EAPI int ecore_evas_rotation_get (Ecore_Evas * *ee*)

Set the rotation of an Ecore_Evas' window.

**Parameters:**

*ee* The Ecore_Evas

**Returns:**

the angle (in degrees) of rotation.

#### 8.6.2.61 EAPI void ecore_evas_rotation_set (Ecore_Evas * *ee*, int *rot*)

Set the rotation of an Ecore_Evas' window.

**Parameters:**

*ee* The Ecore_Evas
*rot* the angle (in degrees) of rotation.

The allowed values of `rot` depend on the engine being used. Most only allow multiples of 90.

#### 8.6.2.62 EAPI int ecore_evas_shaped_get (Ecore_Evas * *ee*)

Query whether an Ecore_Evas is shaped or not.

**Parameters:**

*ee* The Ecore_Evas to query.

**Returns:**

1 if shaped, 0 if not.

This function returns 1 if `ee` is shaped, and 0 if not.

#### 8.6.2.63 EAPI void ecore_evas_shaped_set (Ecore_Evas * *ee*, int *shaped*)

Set whether an Ecore_Evas is shaped or not.

**Parameters:**

***ee*** The Ecore_Evas to shape

***shaped*** 1 to shape, 0 to not

This function allows one to make an Ecore_Evas shaped to the contents of the evas. If `shaped` is 1, `ee` will be transparent in parts of the evas that contain no objects. If `shaped` is 0, then `ee` will be rectangular, and and parts with no data will show random framebuffer artifacting. For non-shaped Ecore_Evases, it is recommend to cover the entire evas with a background object.

#### 8.6.2.64 EAPI void ecore_evas_show (Ecore_Evas * *ee*)

Show an Ecore_Evas' window.

**Parameters:**

***ee*** The Ecore_Evas to show.

This function makes `ee` visible.

#### 8.6.2.65 EAPI int ecore_evas_shutdown (void)

Shut down the Evas system.

**Returns:**

0 if ecore evas is fully shut down, or $> 0$ if it still needs to be shut down

This closes the Evas system down.

#### 8.6.2.66 EAPI void ecore_evas_size_base_get (Ecore_Evas * *ee*, int * *w*, int * *h*)

Get the base size of an Ecore_Evas' window.

**Parameters:**

***ee*** The Ecore_Evas to set

***w*** A pointer to an int to place the base width in.

***h*** A pointer to an int to place the base height in.

This function puts the base size of `ee` into `w` and `h`.

#### 8.6.2.67 EAPI void ecore_evas_size_base_set (Ecore_Evas * *ee*, int *w*, int *h*)

Set the base size of an Ecore_Evas' window.

**Parameters:**

***ee*** The Ecore_Evas to set

***w*** The base width

***h*** The base height

This function sets the base size of `ee` to `w` x `h`.

#### 8.6.2.68 EAPI void ecore_evas_size_max_get (Ecore_Evas * *ee*, int * *w*, int * *h*)

Get the max size of an Ecore_Evas' window.

**Parameters:**

***ee*** The Ecore_Evas to set

***w*** A pointer to an int to place the max width in.

***h*** A pointer to an int to place the max height in.

This function puts the maximum size of `ee` into `w` and `h`.

#### 8.6.2.69 EAPI void ecore_evas_size_max_set (Ecore_Evas * *ee*, int *w*, int *h*)

Set the max size of an Ecore_Evas' window.

**Parameters:**

***ee*** The Ecore_Evas to set

***w*** The maximum width

***h*** The maximum height

This function sets the maximum size of `ee` to `w` x `h`.

#### 8.6.2.70 EAPI void ecore_evas_size_min_get (Ecore_Evas * *ee*, int * *w*, int * *h*)

Get the min size of an Ecore_Evas' window.

**Parameters:**

***ee*** The Ecore_Evas to set

***w*** A pointer to an int to place the min width in.

***h*** A pointer to an int to place the min height in.

This function puts the minimum size of `ee` into `w` and `h`.

#### 8.6.2.71 EAPI void ecore_evas_size_min_set (Ecore_Evas * *ee*, int *w*, int *h*)

Set the min size of an Ecore_Evas' window.

**Parameters:**

***ee*** The Ecore_Evas to set

***w*** The minimum width

***h*** The minimum height

This function sets the minimum size of `ee` to `w` x `h`.

#### 8.6.2.72 EAPI void ecore_evas_size_step_get (Ecore_Evas * *ee*, int * *w*, int * *h*)

Get the step size of an Ecore_Evas' window.

**Parameters:**

***ee*** The Ecore_Evas to set

***w*** A pointer to an int to place the step width in.

***h*** A pointer to an int to place the step height in.

This function puts the step size of `ee` into `w` and `h`.

#### 8.6.2.73 EAPI void ecore_evas_size_step_set (Ecore_Evas * *ee*, int *w*, int *h*)

Set the step size of an Ecore_Evas.

**Parameters:**

***ee*** The Ecore_Evas to set

***w*** The step width

***h*** The step height

This function sets the step size of `ee` to `w` x `h`. This limits the size of an Ecore_Evas to always being an integer multiple of the step size.

#### 8.6.2.74 EAPI int ecore_evas_software_x11_direct_resize_get (Ecore_Evas * *ee*)

To be documented.

FIXME: To be fixed.

#### 8.6.2.75 EAPI void ecore_evas_software_x11_direct_resize_set (Ecore_Evas * *ee*, int *on*)

To be documented.

FIXME: To be fixed.

#### 8.6.2.76 EAPI void ecore_evas_software_x11_extra_event_window_add (Ecore_Evas * *ee*, Ecore_X_Window *win*)

To be documented.

FIXME: To be fixed.

#### 8.6.2.77 EAPI Ecore_Evas* ecore_evas_software_x11_new (const char * *disp_name*, Ecore_X_Window *parent*, int *x*, int *y*, int *w*, int *h*)

To be documented.

FIXME: To be fixed.

#### 8.6.2.78 EAPI Ecore_X_Window ecore_evas_software_x11_subwindow_get (Ecore_Evas * *ee*)

To be documented.

FIXME: To be fixed.

#### 8.6.2.79 EAPI Ecore_X_Window ecore_evas_software_x11_window_get (Ecore_Evas * *ee*)

To be documented.

FIXME: To be fixed.

#### 8.6.2.80 EAPI int ecore_evas_sticky_get (Ecore_Evas * *ee*)

Returns the sticky state of an Ecore_Evas' window.

**Parameters:**

*ee* The Ecore_Evas whose window's sticky state is returned.

**Returns:**

The Ecore_Evas window's sticky state.

#### 8.6.2.81 EAPI void ecore_evas_sticky_set (Ecore_Evas * *ee*, int *sticky*)

Set the sticky state of an Ecore_Evas window.

**Parameters:**

*ee* The Ecore_Evas whose window's sticky state is set.

***sticky*** The Ecore_Evas window's new sticky state.

#### 8.6.2.82 EAPI const char* ecore_evas_title_get (Ecore_Evas * *ee*)

Get the title of an Ecore_Evas' window.

**Parameters:**

*ee* The Ecore_Evas whose title you wish to get.

**Returns:**

The title of `ee`.

This function returns the title of `ee`.

#### 8.6.2.83 EAPI void ecore_evas_title_set (Ecore_Evas * *ee*, const char * *t*)

Set the title of an Ecore_Evas' window.

**Parameters:**

***ee*** The Ecore_Evas whose title you wish to set.
***t*** The title

This function sets the title of `ee` to `t`.

#### 8.6.2.84 EAPI int ecore_evas_visibility_get (Ecore_Evas * *ee*)

Query whether an Ecore_Evas' window is visible or not.

**Parameters:**

***ee*** The Ecore_Evas to query.

**Returns:**

1 if visible, 0 if not.

This function queries `ee` and returns 1 if it is visible, and 0 if not.

#### 8.6.2.85 EAPI int ecore_evas_withdrawn_get (Ecore_Evas * *ee*)

Returns the withdrawn state of an Ecore_Evas' window.

**Parameters:**

***ee*** The Ecore_Evas whose window's withdrawn state is returned.

**Returns:**

The Ecore_Evas window's withdrawn state.

#### 8.6.2.86 EAPI void ecore_evas_withdrawn_set (Ecore_Evas * *ee*, int *withdrawn*)

Set the withdrawn state of an Ecore_Evas' window.

**Parameters:**

***ee*** The Ecore_Evas whose window's withdrawn state is set.
***withdrawn*** The Ecore_Evas window's new withdrawn state.

#### 8.6.2.87 EAPI int ecore_evas_xrender_x11_direct_resize_get (Ecore_Evas * *ee*)

To be documented.

FIXME: To be fixed.

#### 8.6.2.88 EAPI void ecore_evas_xrender_x11_direct_resize_set (Ecore_Evas * *ee*, int *on*)

To be documented.

FIXME: To be fixed.

#### 8.6.2.89 EAPI void ecore_evas_xrender_x11_extra_event_window_add (Ecore_Evas * *ee*, Ecore_X_Window *win*)

To be documented.

FIXME: To be fixed.

#### 8.6.2.90 EAPI Ecore_Evas* ecore_evas_xrender_x11_new (const char * *disp_name*, Ecore_X_Window *parent*, int *x*, int *y*, int *w*, int *h*)

To be documented.

FIXME: To be fixed.

#### 8.6.2.91 EAPI Ecore_X_Window ecore_evas_xrender_x11_subwindow_get (Ecore_Evas * *ee*)

To be documented.

FIXME: To be fixed.

#### 8.6.2.92 EAPI Ecore_X_Window ecore_evas_xrender_x11_window_get (Ecore_Evas * *ee*)

To be documented.

FIXME: To be fixed.

## 8.7 Ecore_Fb.h File Reference

Ecore frame buffer system functions.

### Data Structures

- struct _Ecore_Fb_Event_Key_Down
  *FB Key Down event.*
- struct _Ecore_Fb_Event_Key_Up
  *FB Key Up event.*
- struct _Ecore_Fb_Event_Mouse_Button_Down
  *FB Mouse Down event.*
- struct _Ecore_Fb_Event_Mouse_Button_Up
  *FB Mouse Up event.*
- struct _Ecore_Fb_Event_Mouse_Move
  *FB Mouse Move event.*
- struct _Ecore_Fb_Event_Mouse_Wheel
  *FB Mouse Wheel event.*

### Typedefs

- typedef _Ecore_Fb_Event_Key_Down Ecore_Fb_Event_Key_Down
  *FB Key Down event.*
- typedef _Ecore_Fb_Event_Key_Up Ecore_Fb_Event_Key_Up
  *FB Key Up event.*
- typedef _Ecore_Fb_Event_Mouse_Button_Down Ecore_Fb_Event_Mouse_Button_Down
  *FB Mouse Down event.*
- typedef _Ecore_Fb_Event_Mouse_Button_Up Ecore_Fb_Event_Mouse_Button_Up
  *FB Mouse Up event.*
- typedef _Ecore_Fb_Event_Mouse_Move Ecore_Fb_Event_Mouse_Move
  *FB Mouse Move event.*
- typedef _Ecore_Fb_Event_Mouse_Wheel Ecore_Fb_Event_Mouse_Wheel
  *FB Mouse Wheel event.*

## Functions

- EAPI void ecore_fb_callback_gain_set (void(*func)(void *data), void *data)
  *To be documented.*
- EAPI void ecore_fb_callback_lose_set (void(*func)(void *data), void *data)
  *To be documented.*
- EAPI int ecore_fb_shutdown (void)
  *Shuts down the Ecore_Fb library.*
- EAPI void ecore_fb_size_get (int *w, int *h)
  *Retrieves the width and height of the current frame buffer in pixels.*

## Variables

- EAPI int ECORE_FB_EVENT_KEY_DOWN
  *FB Key Down event.*
- EAPI int ECORE_FB_EVENT_KEY_UP
  *FB Key Up event.*
- EAPI int ECORE_FB_EVENT_MOUSE_BUTTON_DOWN
  *FB Mouse Down event.*
- EAPI int ECORE_FB_EVENT_MOUSE_BUTTON_UP
  *FB Mouse Up event.*
- EAPI int ECORE_FB_EVENT_MOUSE_MOVE
  *FB Mouse Move event.*
- EAPI int ECORE_FB_EVENT_MOUSE_WHEEL
  *FB Mouse Wheel event.*

### 8.7.1 Detailed Description

Ecore frame buffer system functions.

### 8.7.2 Function Documentation

#### 8.7.2.1 EAPI void ecore_fb_callback_gain_set (void(*)(void *data) *func*, void * *data*)

To be documented.

FIXME: To be fixed.

**Todo**

Documentation: Find out what this does.

#### 8.7.2.2 EAPI void ecore_fb_callback_lose_set (void(*)(void *data) *func*, void * *data*)

To be documented.

FIXME: To be fixed.

**Todo**

Documentation: Find out what this does.

#### 8.7.2.3 EAPI void ecore_fb_size_get (int * *w*, int * *h*)

Retrieves the width and height of the current frame buffer in pixels.

**Parameters:**

***w*** Pointer to an integer in which to store the width.

***h*** Pointer to an interge in which to store the height.

## 8.8 Ecore_Ipc.h File Reference

Ecore inter-process communication functions.

### Typedefs

- typedef void Ecore_Ipc_Server
  *An IPC connection handle.*
- typedef void Ecore_Ipc_Client
  *An IPC connection handle.*

### Functions

- EAPI unsigned short _ecore_ipc_swap_16 (unsigned short v)
  *Macros used for generic data packing.*
- EAPI int ecore_ipc_init (void)
  *Initialises the Ecore IPC library.*
- EAPI int ecore_ipc_shutdown (void)
  *Shuts down the Ecore IPC library.*
- EAPI Ecore_Ipc_Server * ecore_ipc_server_add (Ecore_Ipc_Type type, const char *name, int port, const void *data)
  *Creates an IPC server that listens for connections.*
- EAPI Ecore_Ipc_Server * ecore_ipc_server_connect (Ecore_Ipc_Type type, char *name, int port, const void *data)
  *Creates an IPC server object to represent the IPC server listening on the given port.*
- EAPI void * ecore_ipc_server_del (Ecore_Ipc_Server *svr)
  *Closes the connection and frees the given IPC server.*
- EAPI void * ecore_ipc_server_data_get (Ecore_Ipc_Server *svr)
  *Retrieves the data associated with the given IPC server.*
- EAPI int ecore_ipc_server_connected_get (Ecore_Ipc_Server *svr)
  *Retrieves whether the given IPC server is currently connected.*
- EAPI Ecore_List * ecore_ipc_server_clients_get (Ecore_Ipc_Server *svr)
  *Retrieves the list of clients for this server.*
- EAPI int ecore_ipc_server_send (Ecore_Ipc_Server *svr, int major, int minor, int ref, int ref_to, int response, void *data, int size)

*Sends a message to the given IPC server.*

- EAPI void ecore_ipc_server_client_limit_set (Ecore_Ipc_Server *svr, int client_limit, char reject_excess_clients)

  *Sets a limit on the number of clients that can be handled concurrently by the given server, and a policy on what to do if excess clients try to connect.*

- EAPI void ecore_ipc_server_data_size_max_set (Ecore_Ipc_Server *srv, int size)

  *Sets the max data payload size for an Ipc message in bytes.*

- EAPI int ecore_ipc_server_data_size_max_get (Ecore_Ipc_Server *srv)

  *Gets the max data payload size for an Ipc message in bytes.*

- EAPI char * ecore_ipc_server_ip_get (Ecore_Ipc_Server *svr)

  *Gets the IP address of a server that has been connected to.*

- EAPI void ecore_ipc_server_flush (Ecore_Ipc_Server *svr)

  *Flushes all pending data to the given server.*

- EAPI int ecore_ipc_client_send (Ecore_Ipc_Client *cl, int major, int minor, int ref, int ref_to, int response, void *data, int size)

  *Sends a message to the given IPC client.*

- EAPI Ecore_Ipc_Server * ecore_ipc_client_server_get (Ecore_Ipc_Client *cl)

  *Retrieves the IPC server that the given IPC client is connected to.*

- EAPI void * ecore_ipc_client_del (Ecore_Ipc_Client *cl)

  *Closes the connection and frees memory allocated to the given IPC client.*

- EAPI void ecore_ipc_client_data_set (Ecore_Ipc_Client *cl, const void *data)

  *Sets the IPC data associated with the given IPC client to* `data`.

- EAPI void * ecore_ipc_client_data_get (Ecore_Ipc_Client *cl)

  *Retrieves the data that has been associated with the given IPC client.*

- EAPI void ecore_ipc_client_data_size_max_set (Ecore_Ipc_Client *cl, int size)

  *Sets the max data payload size for an Ipc message in bytes.*

- EAPI int ecore_ipc_client_data_size_max_get (Ecore_Ipc_Client *cl)

  *Sets the max data payload size for an Ipc message in bytes.*

- EAPI char * ecore_ipc_client_ip_get (Ecore_Ipc_Client *cl)

  *Gets the IP address of a client that has been connected to.*

- EAPI void ecore_ipc_client_flush (Ecore_Ipc_Client *cl)

  *Flushes all pending data to the given client.*

- EAPI int ecore_ipc_ssl_available_get (void)

  *Returns if SSL support is available.*

### 8.8.1 Detailed Description

Ecore inter-process communication functions.

### 8.8.2 Function Documentation

#### 8.8.2.1 EAPI int ecore_ipc_client_data_size_max_get (Ecore_Ipc_Client * *cl*)

Sets the max data payload size for an Ipc message in bytes.

**Parameters:**

***cl*** The given client.

***size*** The maximum data payload size in bytes.

#### 8.8.2.2 EAPI void ecore_ipc_client_data_size_max_set (Ecore_Ipc_Client * *cl*, int *size*)

Sets the max data payload size for an Ipc message in bytes.

**Parameters:**

***client*** The given client.

***size*** The maximum data payload size in bytes.

#### 8.8.2.3 EAPI void ecore_ipc_client_flush (Ecore_Ipc_Client * *cl*)

Flushes all pending data to the given client.

Will return when done.

**Parameters:**

***cl*** The given client.

#### 8.8.2.4 EAPI char* ecore_ipc_client_ip_get (Ecore_Ipc_Client * *cl*)

Gets the IP address of a client that has been connected to.

**Parameters:**

***cl*** The given client.

**Returns:**

A pointer to an internal string that contains the IP address of the connected server in the form "XXX.YYY.ZZZ.AAA" IP notation. This string should not be modified or trusted to stay valid after deletion for the `cl` object. If no IP is known NULL is returned.

#### 8.8.2.5 EAPI void ecore_ipc_server_client_limit_set (Ecore_Ipc_Server * *svr*, int *client_limit*, char *reject_excess_clients*)

Sets a limit on the number of clients that can be handled concurrently by the given server, and a policy on what to do if excess clients try to connect.

Beware that if you set this once ecore is already running, you may already have pending CLIENT_ADD events in your event queue. Those clients have already connected and will not be affected by this call. Only clients subsequently trying to connect will be affected.

**Parameters:**

***svr*** The given server.

***client_limit*** The maximum number of clients to handle concurrently. -1 means unlimited (default). 0 effectively disables the server.

***reject_excess_clients*** Set to 1 to automatically disconnect excess clients as soon as they connect if you are already handling client_limit clients. Set to 0 (default) to just hold off on the "accept()" system call until the number of active clients drops. This causes the kernel to queue up to 4096 connections (or your kernel's limit, whichever is lower).

#### 8.8.2.6 EAPI int ecore_ipc_server_data_size_max_get (Ecore_Ipc_Server * *svr*)

Gets the max data payload size for an Ipc message in bytes.

**Parameters:**

***svr*** The given server.

**Returns:**

The maximum data payload in bytes.

#### 8.8.2.7 EAPI void ecore_ipc_server_data_size_max_set (Ecore_Ipc_Server * *svr*, int *size*)

Sets the max data payload size for an Ipc message in bytes.

**Parameters:**

***svr*** The given server.

***size*** The maximum data payload size in bytes.

#### 8.8.2.8 EAPI void ecore_ipc_server_flush (Ecore_Ipc_Server * *svr*)

Flushes all pending data to the given server.

Will return when done.

**Parameters:**

***svr*** The given server.

#### 8.8.2.9 EAPI char* ecore_ipc_server_ip_get (Ecore_Ipc_Server * *svr*)

Gets the IP address of a server that has been connected to.

**Parameters:**

***svr*** The given server.

**Returns:**

A pointer to an internal string that contains the IP address of the connected server in the form "XXX.YYY.ZZZ.AAA" IP notation. This string should not be modified or trusted to stay valid after deletion for the `svr` object. If no IP is known NULL is returned.

## 8.9 Ecore_Job.h File Reference

Functions for dealing with Ecore jobs.

### Typedefs

- typedef void Ecore_Job
  *A job handle.*

### Functions

- EAPI Ecore_Job * ecore_job_add (void(*func)(void *data), const void *data)
  *Add a job to the event queue.*
- EAPI void * ecore_job_del (Ecore_Job *job)
  *Delete a queued job that has not yet been executed.*

### 8.9.1 Detailed Description

Functions for dealing with Ecore jobs.

## 8.10 Ecore_Str.h File Reference

Contains useful C string functions.

### Functions

- EAPI char ** ecore_str_split (const char *string, const char *delimiter, int max_tokens)
  *Splits a string into a maximum of max_tokens pieces, using the given delimiter.*

### 8.10.1 Detailed Description

Contains useful C string functions.

### 8.10.2 Function Documentation

#### 8.10.2.1 EAPI char** ecore_str_split (const char * *str*, const char * *delim*, int *max_tokens*)

Splits a string into a maximum of max_tokens pieces, using the given delimiter.

If max_tokens is reached, the final string in the returned string array contains the remainder of string.

**Parameters:**

***str*** A string to split.

***delim*** A string which specifies the places at which to split the string. The delimiter is not included in any of the resulting strings, unless max_tokens is reached.

***max_tokens*** The maximum number of strings to split string into. If this is less than 1, the string is split completely.

**Returns:**

A newly-allocated NULL-terminated array of strings. To free it: free the first element of the array and the array itself.

## 8.11 Ecore_Txt.h File Reference

Provides a text encoding conversion function.

### Functions

- EAPI char * ecore_txt_convert (const char *enc_from, const char *enc_to, const char *text)

  *To be documented.*

### 8.11.1 Detailed Description

Provides a text encoding conversion function.

### 8.11.2 Function Documentation

#### 8.11.2.1 EAPI char* ecore_txt_convert (const char * *enc_from*, const char * *enc_to*, const char * *text*)

To be documented.

FIXME: Finish this.

## 8.12 Ecore_X.h File Reference

Ecore functions for dealing with the X Windows System.

### Enumerations

- enum _Ecore_X_Window_State {
  ECORE_X_WINDOW_STATE_ICONIFIED,
  ECORE_X_WINDOW_STATE_MODAL,
  ECORE_X_WINDOW_STATE_STICKY,
  ECORE_X_WINDOW_STATE_MAXIMIZED_VERT,
  ECORE_X_WINDOW_STATE_MAXIMIZED_HORZ,
  ECORE_X_WINDOW_STATE_SHADED,
  ECORE_X_WINDOW_STATE_SKIP_TASKBAR,
  ECORE_X_WINDOW_STATE_SKIP_PAGER,
  ECORE_X_WINDOW_STATE_HIDDEN,
  ECORE_X_WINDOW_STATE_FULLSCREEN }
- enum _Ecore_X_WM_Protocol {
  ECORE_X_WM_PROTOCOL_DELETE_REQUEST,
  ECORE_X_WM_PROTOCOL_TAKE_FOCUS,
  ECORE_X_NET_WM_PROTOCOL_PING,
  ECORE_X_NET_WM_PROTOCOL_SYNC_REQUEST }
- enum _Ecore_X_Window_Input_Mode {
  ECORE_X_WINDOW_INPUT_MODE_NONE,
  ECORE_X_WINDOW_INPUT_MODE_PASSIVE,
  ECORE_X_WINDOW_INPUT_MODE_ACTIVE_LOCAL,
  ECORE_X_WINDOW_INPUT_MODE_ACTIVE_GLOBAL }
- enum _Ecore_X_Window_State_Hint {
  ECORE_X_WINDOW_STATE_HINT_NONE = -1,
  ECORE_X_WINDOW_STATE_HINT_WITHDRAWN,
  ECORE_X_WINDOW_STATE_HINT_NORMAL,
  ECORE_X_WINDOW_STATE_HINT_ICONIC }

### Functions

- EAPI int ecore_x_init (const char *name)

  *Initialize the X display connection to the given display.*

- EAPI int ecore_x_shutdown (void)

  *Shuts down the Ecore X library.*

- EAPI int ecore_x_disconnect (void)

  *Shuts down the Ecore X library.*

- EAPI Ecore_X_Display * ecore_x_display_get (void)

  *Retrieves the Ecore_X_Display handle used for the current X connection.*

- EAPI int ecore_x_fd_get (void)

  *Retrieves the X display file descriptor.*

- EAPI void ecore_x_double_click_time_set (double t)

  *Sets the timeout for a double and triple clicks to be flagged.*

- EAPI double ecore_x_double_click_time_get (void)

  *Retrieves the double and triple click flag timeout.*

- EAPI void ecore_x_flush (void)

  *Sends all X commands in the X Display buffer.*

- EAPI void ecore_x_sync (void)

  *Flushes the command buffer and waits until all requests have been processed by the server.*

- EAPI void ecore_x_killall (Ecore_X_Window root)

  *Kill all clients with subwindows under a given window.*

- EAPI void ecore_x_kill (Ecore_X_Window win)

  *Kill a specific client.*

- EAPI Ecore_X_Time ecore_x_current_time_get (void)

  *Return the last event time.*

- EAPI void ecore_x_error_handler_set (void(*func)(void *data), const void *data)

  *Set the error handler.*

- EAPI void ecore_x_io_error_handler_set (void(*func)(void *data), const void *data)

  *Set the I/O error handler.*

- EAPI int ecore_x_error_request_get (void)

  *Get the request code that caused the error.*

- EAPI int ecore_x_error_code_get (void)

  *Get the error code from the error.*

- EAPI int ecore_x_selection_primary_set (Ecore_X_Window w, const void *data, int size)

  *Claim ownership of the PRIMARY selection and set its data.*

- EAPI int ecore_x_selection_primary_clear (void)

  *Release ownership of the primary selection.*

- EAPI int ecore_x_selection_secondary_set (Ecore_X_Window w, const void *data, int size)

  *Claim ownership of the SECONDARY selection and set its data.*

- EAPI int ecore_x_selection_secondary_clear (void)

  *Release ownership of the secondary selection.*

- EAPI int ecore_x_selection_xdnd_set (Ecore_X_Window w, const void *data, int size)

  *Claim ownership of the XDND selection and set its data.*

- EAPI int ecore_x_selection_xdnd_clear (void)

  *Release ownership of the XDND selection.*

- EAPI int ecore_x_selection_clipboard_set (Ecore_X_Window w, const void *data, int size)

  *Claim ownership of the CLIPBOARD selection and set its data.*

- EAPI int ecore_x_selection_clipboard_clear (void)

  *Release ownership of the clipboard selection.*

- EAPI Ecore_X_Window ecore_x_window_new (Ecore_X_Window parent, int x, int y, int w, int h)

  *Creates a new window.*

- EAPI Ecore_X_Window ecore_x_window_override_new (Ecore_X_Window parent, int x, int y, int w, int h)

  *Creates a window with the override redirect attribute set to* `True`.

- EAPI Ecore_X_Window ecore_x_window_manager_argb_new (Ecore_X_Window parent, int x, int y, int w, int h)

  *Creates a new window.*

- EAPI Ecore_X_Window ecore_x_window_argb_new (Ecore_X_Window parent, int x, int y, int w, int h)

  *Creates a new window.*

- EAPI Ecore_X_Window ecore_x_window_override_argb_new (Ecore_X_Window parent, int x, int y, int w, int h)

  *Creates a window with the override redirect attribute set to* `True`.

- EAPI Ecore_X_Window ecore_x_window_input_new (Ecore_X_Window parent, int x, int y, int w, int h)

  *Creates a new input window.*

- EAPI void ecore_x_window_del (Ecore_X_Window win)

  *Deletes the given window.*

- EAPI void ecore_x_window_ignore_set (Ecore_X_Window win, int ignore)

  *Set if a window should be ignored.*

- EAPI Ecore_X_Window * ecore_x_window_ignore_list (int *num)

  *Get the ignore list.*

- EAPI void ecore_x_window_delete_request_send (Ecore_X_Window win)

  *Sends a delete request to the given window.*

- EAPI void ecore_x_window_show (Ecore_X_Window win)

  *Shows a window.*

- EAPI void ecore_x_window_hide (Ecore_X_Window win)

  *Hides a window.*

- EAPI void ecore_x_window_move (Ecore_X_Window win, int x, int y)

  *Moves a window to the position* `x`, `y`.

- EAPI void ecore_x_window_resize (Ecore_X_Window win, int w, int h)

  *Resizes a window.*

- EAPI void ecore_x_window_move_resize (Ecore_X_Window win, int x, int y, int w, int h)

  *Moves and resizes a window.*

- EAPI void ecore_x_window_focus (Ecore_X_Window win)

  *Sets the focus to the window* `win`.

- EAPI void ecore_x_window_focus_at_time (Ecore_X_Window win, Ecore_X_Time t)

  *Sets the focus to the given window at a specific time.*

- EAPI Ecore_X_Window ecore_x_window_focus_get (void)

  *gets the focus to the window* `win`.

- EAPI void ecore_x_window_raise (Ecore_X_Window win)

  *Raises the given window.*

- EAPI void ecore_x_window_lower (Ecore_X_Window win)

  *Lowers the given window.*

- EAPI void ecore_x_window_reparent (Ecore_X_Window win, Ecore_X_Window new_parent, int x, int y)

  *Moves a window to within another window at a given position.*

- EAPI void ecore_x_window_size_get (Ecore_X_Window win, int *w, int *h)

  *Retrieves the size of the given window.*

- EAPI void ecore_x_window_geometry_get (Ecore_X_Window win, int *x, int *y, int *w, int *h)

  *Retrieves the geometry of the given window.*

- EAPI int ecore_x_window_border_width_get (Ecore_X_Window win)

*Retrieves the width of the border of the given window.*

- EAPI void ecore_x_window_border_width_set (Ecore_X_Window win, int width)

  *Sets the width of the border of the given window.*

- EAPI int ecore_x_window_depth_get (Ecore_X_Window win)

  *Retrieves the depth of the given window.*

- EAPI void ecore_x_window_cursor_show (Ecore_X_Window win, int show)

  *To be documented.*

- EAPI void ecore_x_window_defaults_set (Ecore_X_Window win)

  *Sets the default properties for the given window.*

- EAPI int ecore_x_window_visible_get (Ecore_X_Window win)

  *Finds out whether the given window is currently visible.*

- EAPI Ecore_X_Window ecore_x_window_at_xy_get (int x, int y)

  *Retrieves the top, visible window at the given location.*

- EAPI Ecore_X_Window ecore_x_window_at_xy_with_skip_get (int x, int y, Ecore_X_Window *skip, int skip_num)

  *Retrieves the top, visible window at the given location, but skips the windows in the list.*

- EAPI Ecore_X_Window ecore_x_window_parent_get (Ecore_X_Window win)

  *Retrieves the parent window of the given window.*

- EAPI void ecore_x_window_background_color_set (Ecore_X_Window win, unsigned short r, unsigned short g, unsigned short b)

  *Sets the background color of the given window.*

- EAPI Ecore_X_Atom ecore_x_window_prop_any_type (void)

  *To be documented.*

- EAPI void ecore_x_window_prop_property_set (Ecore_X_Window win, Ecore_X_Atom type, Ecore_X_Atom format, int size, void *data, int number)

  *To be documented.*

- EAPI int ecore_x_window_prop_property_get (Ecore_X_Window win, Ecore_X_Atom property, Ecore_X_Atom type, int size, unsigned char **data, int *num)

  *To be documented.*

- EAPI void ecore_x_window_prop_string_set (Ecore_X_Window win, Ecore_X_Atom type, const char *str)

  *Set a window string property.*

- EAPI char * ecore_x_window_prop_string_get (Ecore_X_Window win, Ecore_X_Atom type)

  *Get a window string property.*

- EAPI Ecore_X_WM_Protocol * ecore_x_window_prop_protocol_list_get (Ecore_X_Window win, int *num_ret)

  *To be documented.*

- EAPI void ecore_x_window_shape_mask_set (Ecore_X_Window win, Ecore_X_Pixmap mask)

  *Sets the shape of the given window to that given by the pixmap* `mask`.

- EAPI Ecore_X_Pixmap ecore_x_pixmap_new (Ecore_X_Window win, int w, int h, int dep)

  *Creates a new pixmap.*

- EAPI void ecore_x_pixmap_del (Ecore_X_Pixmap pmap)

  *Deletes the reference to the given pixmap.*

- EAPI void ecore_x_pixmap_paste (Ecore_X_Pixmap pmap, Ecore_X_Drawable dest, Ecore_X_GC gc, int sx, int sy, int w, int h, int dx, int dy)

  *Pastes a rectangular area of the given pixmap onto the given drawable.*

- EAPI void ecore_x_pixmap_geometry_get (Ecore_X_Pixmap pmap, int *x, int *y, int *w, int *h)

  *Retrieves the size of the given pixmap.*

- EAPI int ecore_x_pixmap_depth_get (Ecore_X_Pixmap pmap)

  *Retrieves the depth of the given pixmap.*

- EAPI Ecore_X_GC ecore_x_gc_new (Ecore_X_Drawable draw)

  *Creates a new default graphics context associated with the given drawable.*

- EAPI void ecore_x_gc_del (Ecore_X_GC gc)

  *Deletes and frees the given graphics context.*

- EAPI int ecore_x_client_message32_send (Ecore_X_Window win, Ecore_X_Atom type, Ecore_X_Event_Mask mask, long d0, long d1, long d2, long d3, long d4)

  *Send client message with given type and format 32.*

- EAPI int ecore_x_client_message8_send (Ecore_X_Window win, Ecore_X_Atom type, const void *data, int len)

  *Send client message with given type and format 8.*

- EAPI void ecore_x_drawable_geometry_get (Ecore_X_Drawable d, int *x, int *y, int *w, int *h)

  *Retrieves the geometry of the given drawable.*

- EAPI int ecore_x_drawable_border_width_get (Ecore_X_Drawable d)

  *Retrieves the width of the border of the given drawable.*

- EAPI int ecore_x_drawable_depth_get (Ecore_X_Drawable d)

  *Retrieves the depth of the given drawable.*

- EAPI Ecore_X_Window * ecore_x_window_root_list (int *num_ret)

  *Get a list of all the root windows on the server.*

- EAPI Ecore_X_Atom ecore_x_atom_get (const char *name)

  *Retrieves the atom value associated with the given name.*

- EAPI void ecore_x_icccm_protocol_set (Ecore_X_Window win, Ecore_X_WM_Protocol protocol, int on)

  *Set or unset a wm protocol property.*

- EAPI int ecore_x_icccm_protocol_isset (Ecore_X_Window win, Ecore_X_WM_Protocol protocol)

  *Determines whether a protocol is set for a window.*

- EAPI void ecore_x_icccm_name_class_set (Ecore_X_Window win, const char *n, const char *c)

  *Set a window name & class.*

- EAPI void ecore_x_icccm_name_class_get (Ecore_X_Window win, char **n, char **c)

  *Get a window name & class.*

- EAPI char * ecore_x_icccm_client_machine_get (Ecore_X_Window win)

  *Get a window client machine string.*

- EAPI void ecore_x_icccm_command_set (Ecore_X_Window win, int argc, char **argv)

  *Sets the WM_COMMAND property for* win.

- EAPI void ecore_x_icccm_command_get (Ecore_X_Window win, int *argc, char ***argv)

  *Get the WM_COMMAND property for* win.

- EAPI char * ecore_x_icccm_icon_name_get (Ecore_X_Window win)

  *Get a window icon name.*

- EAPI void ecore_x_icccm_icon_name_set (Ecore_X_Window win, const char *t)

  *Set a window icon name.*

- EAPI void ecore_x_icccm_colormap_window_set (Ecore_X_Window win, Ecore_X_Window subwin)

  *Add a subwindow to the list of windows that need a different colormap installed.*

- EAPI void ecore_x_icccm_colormap_window_unset (Ecore_X_Window win, Ecore_X_Window subwin)

  *Remove a window from the list of colormap windows.*

- EAPI void ecore_x_icccm_transient_for_set (Ecore_X_Window win, Ecore_X_Window forwin)

  *Specify that a window is transient for another top-level window and should be handled accordingly.*

- EAPI void ecore_x_icccm_transient_for_unset (Ecore_X_Window win)

*Remove the transient_for setting from a window.*

- EAPI Ecore_X_Window ecore_x_icccm_transient_for_get (Ecore_X_Window win)

  *Get the window this window is transient for, if any.*

- EAPI void ecore_x_icccm_window_role_set (Ecore_X_Window win, const char *role)

  *Set the window role hint.*

- EAPI char * ecore_x_icccm_window_role_get (Ecore_X_Window win)

  *Get the window role.*

- EAPI void ecore_x_icccm_client_leader_set (Ecore_X_Window win, Ecore_X_Window l)

  *Set the window's client leader.*

- EAPI Ecore_X_Window ecore_x_icccm_client_leader_get (Ecore_X_Window win)

  *Get the window's client leader.*

- EAPI int ecore_x_dpms_query (void)

  *Checks if the X DPMS extension is available on the server.*

- EAPI int ecore_x_dpms_capable_get (void)

  *Checks if the X server is capable of DPMS.*

- EAPI int ecore_x_dpms_enabled_get (void)

  *Checks the DPMS state of the display.*

- EAPI void ecore_x_dpms_enabled_set (int enabled)

  *Sets the DPMS state of the display.*

- EAPI void ecore_x_dpms_timeouts_get (unsigned int *standby, unsigned int *suspend, unsigned int *off)

  *Gets the timeouts.*

- EAPI int ecore_x_dpms_timeouts_set (unsigned int standby, unsigned int suspend, unsigned int off)

  *Sets the timeouts.*

- EAPI unsigned int ecore_x_dpms_timeout_standby_get ()

  *Returns the amount of time of inactivity before standby mode is invoked.*

- EAPI unsigned int ecore_x_dpms_timeout_suspend_get ()

  *Returns the amount of time of inactivity before the second level of power saving is invoked.*

- EAPI unsigned int ecore_x_dpms_timeout_off_get ()

  *Returns the amount of time of inactivity before the third and final level of power saving is invoked.*

- EAPI void ecore_x_dpms_timeout_standby_set (unsigned int new_timeout)

  *Sets the standby timeout (in unit of seconds).*

- EAPI void ecore_x_dpms_timeout_suspend_set (unsigned int new_timeout)
  *Sets the suspend timeout (in unit of seconds).*
- EAPI void ecore_x_dpms_timeout_off_set (unsigned int new_timeout)
  *Sets the off timeout (in unit of seconds).*

## 8.12.1 Detailed Description

Ecore functions for dealing with the X Windows System.

Ecore_X provides a wrapper and convenience functions for using the X Windows System. Function groups for this part of the library include the following:

- X Library Init and Shutdown Functions
- X Display Attributes
- X Synchronization Functions

## 8.12.2 Enumeration Type Documentation

### 8.12.2.1 enum _Ecore_X_Window_Input_Mode

**Enumerator:**

***ECORE_X_WINDOW_INPUT_MODE_NONE*** The window can never be focused.

***ECORE_X_WINDOW_INPUT_MODE_PASSIVE*** The window can be focused by the WM but doesn't focus itself.

***ECORE_X_WINDOW_INPUT_MODE_ACTIVE_LOCAL*** The window sets the focus itself if one of its sub-windows already is focused.

***ECORE_X_WINDOW_INPUT_MODE_ACTIVE_GLOBAL*** The window sets the focus itself even if another window is currently focused.

### 8.12.2.2 enum _Ecore_X_Window_State

**Enumerator:**

***ECORE_X_WINDOW_STATE_ICONIFIED*** The window is iconified.

***ECORE_X_WINDOW_STATE_MODAL*** The window is a modal dialog box.

***ECORE_X_WINDOW_STATE_STICKY*** The window manager should keep the window's position fixed even if the virtual desktop scrolls.

***ECORE_X_WINDOW_STATE_MAXIMIZED_VERT*** The window has the maximum vertical size.

***ECORE_X_WINDOW_STATE_MAXIMIZED_HORZ*** The window has the maximum horizontal size.

***ECORE_X_WINDOW_STATE_SHADED*** The window is shaded.

***ECORE_X_WINDOW_STATE_SKIP_TASKBAR*** The window should not be included in the taskbar.

***ECORE_X_WINDOW_STATE_SKIP_PAGER*** The window should not be included in the pager.

***ECORE_X_WINDOW_STATE_HIDDEN*** The window is invisible (i.e. minimized/iconified)

***ECORE_X_WINDOW_STATE_FULLSCREEN*** The window should fill the entire screen and have no window border/decorations.

#### 8.12.2.3 enum _Ecore_X_Window_State_Hint

**Enumerator:**

***ECORE_X_WINDOW_STATE_HINT_NONE*** Do not provide any state hint to the window manager.

***ECORE_X_WINDOW_STATE_HINT_WITHDRAWN*** The window wants to remain hidden and NOT iconified.

***ECORE_X_WINDOW_STATE_HINT_NORMAL*** The window wants to be mapped normally.

***ECORE_X_WINDOW_STATE_HINT_ICONIC*** The window wants to start in an iconified state.

#### 8.12.2.4 enum _Ecore_X_WM_Protocol

**Enumerator:**

***ECORE_X_WM_PROTOCOL_DELETE_REQUEST*** If enabled the window manager will be asked to send a delete message instead of just closing (destroying) the window.

***ECORE_X_WM_PROTOCOL_TAKE_FOCUS*** If enabled the window manager will be told that the window explicitly sets input focus.

***ECORE_X_NET_WM_PROTOCOL_PING*** If enabled the window manager can ping the window to check if it is alive.

***ECORE_X_NET_WM_PROTOCOL_SYNC_REQUEST*** If enabled the window manager can sync updating with the window (?).

### 8.12.3 Function Documentation

#### 8.12.3.1 EAPI Ecore_X_Atom ecore_x_atom_get (const char * *name*)

Retrieves the atom value associated with the given name.

**Parameters:**

***name*** The given name.

**Returns:**

Associated atom value.

#### 8.12.3.2 EAPI int ecore_x_client_message32_send (Ecore_X_Window *win*, Ecore_X_Atom *type*, Ecore_X_Event_Mask *mask*, long *d0*, long *d1*, long *d2*, long *d3*, long *d4*)

Send client message with given type and format 32.

**Parameters:**

***win*** The window the message is sent to.

***type*** The client message type.

***d0*** The client message data item 1

***d1*** The client message data item 2

***d2*** The client message data item 3

***d3*** The client message data item 4

***d4*** The client message data item 5

**Returns:**

!0 on success.

#### 8.12.3.3 EAPI int ecore_x_client_message8_send (Ecore_X_Window *win*, Ecore_X_Atom *type*, const void * *data*, int *len*)

Send client message with given type and format 8.

**Parameters:**

***win*** The window the message is sent to.

***type*** The client message type.

***data*** Data to be sent.

***len*** Number of data bytes, max 20.

**Returns:**

!0 on success.

#### 8.12.3.4 EAPI int ecore_x_error_code_get (void)

Get the error code from the error.

**Returns:**

The error code from the X error

Return the error code from the last X error

#### 8.12.3.5 EAPI void ecore_x_error_handler_set (void(*)(void *data) *func*, const void * *data*)

Set the error handler.

**Parameters:**

***func*** The error handler function

***data*** The data to be passed to the handler function

Set the X error handler function

#### 8.12.3.6 EAPI int ecore_x_error_request_get (void)

Get the request code that caused the error.

**Returns:**

The request code causing the X error

Return the X request code that caused the last X error

#### 8.12.3.7 EAPI void ecore_x_gc_del (Ecore_X_GC *gc*)

Deletes and frees the given graphics context.

**Parameters:**

***gc*** The given graphics context.

#### 8.12.3.8 EAPI Ecore_X_GC ecore_x_gc_new (Ecore_X_Drawable *draw*)

Creates a new default graphics context associated with the given drawable.

**Parameters:**

***draw*** Drawable to create graphics context with. If 0 is given instead, the default root window is used.

**Returns:**

The new default graphics context.

#### 8.12.3.9 EAPI Ecore_X_Window ecore_x_icccm_client_leader_get (Ecore_X_Window *win*)

Get the window's client leader.

**Parameters:**

***win*** The window

**Returns:**

The window's client leader window, or 0 if unset

#### 8.12.3.10 EAPI void ecore_x_icccm_client_leader_set (Ecore_X_Window *win*, Ecore_X_Window *l*)

Set the window's client leader.

**Parameters:**

***win*** The window

***l*** The client leader window

All non-transient top-level windows created by an app other than the main window must have this property set to the app's main window.

#### 8.12.3.11 EAPI char* ecore_x_icccm_client_machine_get (Ecore_X_Window *win*)

Get a window client machine string.

**Parameters:**

***win*** The window

**Returns:**

The windows client machine string

Return the client machine of a window. String must be free'd when done with.

#### 8.12.3.12 EAPI void ecore_x_icccm_colormap_window_set (Ecore_X_Window *win*, Ecore_X_Window *subwin*)

Add a subwindow to the list of windows that need a different colormap installed.

**Parameters:**

***win*** The toplevel window

***subwin*** The subwindow to be added to the colormap windows list

#### 8.12.3.13 EAPI void ecore_x_icccm_colormap_window_unset (Ecore_X_Window *win*, Ecore_X_Window *subwin*)

Remove a window from the list of colormap windows.

**Parameters:**

***win*** The toplevel window

***subwin*** The window to be removed from the colormap window list.

#### 8.12.3.14 EAPI void ecore_x_icccm_command_get (Ecore_X_Window *win*, int * *argc*, char *** *argv*)

Get the WM_COMMAND property for *win*.

Return the command of a window. String must be free'd when done with.

**Parameters:**

***win*** The window.

***argc*** Number of arguments.

***argv*** Arguments.

#### 8.12.3.15 EAPI void ecore_x_icccm_command_set (Ecore_X_Window *win*, int *argc*, char ** *argv*)

Sets the WM_COMMAND property for *win*.

**Parameters:**

***win*** The window.

***argc*** Number of arguments.

***argv*** Arguments.

#### 8.12.3.16 EAPI char* ecore_x_icccm_icon_name_get (Ecore_X_Window *win*)

Get a window icon name.

**Parameters:**

***win*** The window

**Returns:**

The windows icon name string

Return the icon name of a window. String must be free'd when done with.

#### 8.12.3.17 EAPI void ecore_x_icccm_icon_name_set (Ecore_X_Window *win*, const char * *t*)

Set a window icon name.

**Parameters:**

***win*** The window

***t*** The icon name string

Set a window icon name

#### 8.12.3.18 EAPI void ecore_x_icccm_name_class_get (Ecore_X_Window *win*, char ** *n*, char ** *c*)

Get a window name & class.

**Parameters:**

***win*** The window

***n*** The name string

***c*** The class string

Get a window name * class

#### 8.12.3.19 EAPI void ecore_x_icccm_name_class_set (Ecore_X_Window *win*, const char * *n*, const char * *c*)

Set a window name & class.

**Parameters:**

***win*** The window

***n*** The name string

***c*** The class string

Set a window name * class

#### 8.12.3.20 EAPI int ecore_x_icccm_protocol_isset (Ecore_X_Window *win*, Ecore_X_WM_Protocol *protocol*)

Determines whether a protocol is set for a window.

**Parameters:**

***win*** The Window

***protocol*** The protocol to query

**Returns:**

1 if the protocol is set, else 0.

#### 8.12.3.21 EAPI void ecore_x_icccm_protocol_set (Ecore_X_Window *win*, Ecore_X_WM_Protocol *protocol*, int *on*)

Set or unset a wm protocol property.

**Parameters:**

***win*** The Window

***protocol*** The protocol to enable/disable

***on*** On/Off

#### 8.12.3.22 EAPI Ecore_X_Window ecore_x_icccm_transient_for_get (Ecore_X_Window *win*)

Get the window this window is transient for, if any.

**Parameters:**

***win*** The window to check

**Returns:**

The window ID of the top-level window, or 0 if the property does not exist.

#### 8.12.3.23 EAPI void ecore_x_icccm_transient_for_set (Ecore_X_Window *win*, Ecore_X_Window *forwin*)

Specify that a window is transient for another top-level window and should be handled accordingly.

**Parameters:**

***win*** the transient window

***forwin*** the toplevel window

#### 8.12.3.24 EAPI void ecore_x_icccm_transient_for_unset (Ecore_X_Window *win*)

Remove the transient_for setting from a window.

**Parameters:**

***The*** window

#### 8.12.3.25 EAPI char* ecore_x_icccm_window_role_get (Ecore_X_Window *win*)

Get the window role.

**Parameters:**

***win*** The window

**Returns:**

The window's role string.

#### 8.12.3.26 EAPI void ecore_x_icccm_window_role_set (Ecore_X_Window *win*, const char * *role*)

Set the window role hint.

**Parameters:**

***win*** The window

***role*** The role string

#### 8.12.3.27 EAPI void ecore_x_io_error_handler_set (void(*)(void *data) *func*, const void * *data*)

Set the I/O error handler.

**Parameters:**

***func*** The I/O error handler function

***data*** The data to be passed to the handler function

Set the X I/O error handler function

#### 8.12.3.28 EAPI void ecore_x_kill (Ecore_X_Window *win*)

Kill a specific client.

You can kill a specific client woning window `win`

**Parameters:**

***win*** Window of the client to be killed

#### 8.12.3.29 EAPI void ecore_x_killall (Ecore_X_Window *root*)

Kill all clients with subwindows under a given window.

You can kill all clients connected to the X server by using ecore_x_window_root_list to get a list of root windows, and then passing each root window to this function.

**Parameters:**

***root*** The window whose children will be killed.

#### 8.12.3.30 EAPI int ecore_x_selection_clipboard_clear (void)

Release ownership of the clipboard selection.

**Returns:**

Returns 1 if the selection was successfully cleared, or 0 if unsuccessful.

#### 8.12.3.31 EAPI int ecore_x_selection_clipboard_set (Ecore_X_Window *w*, const void * *data*, int *size*)

Claim ownership of the CLIPBOARD selection and set its data.

**Parameters:**

***w*** The window to which this selection belongs

***data*** The data associated with the selection

***size*** The size of the data buffer in bytes

**Returns:**

Returns 1 if the ownership of the selection was successfully claimed, or 0 if unsuccessful.

Get the converted data from a previous CLIPBOARD selection request. The buffer must be freed when done with.

#### 8.12.3.32 EAPI int ecore_x_selection_primary_clear (void)

Release ownership of the primary selection.

**Returns:**

Returns 1 if the selection was successfully cleared, or 0 if unsuccessful.

#### 8.12.3.33 EAPI int ecore_x_selection_primary_set (Ecore_X_Window *w*, const void * *data*, int *size*)

Claim ownership of the PRIMARY selection and set its data.

**Parameters:**

***w*** The window to which this selection belongs

***data*** The data associated with the selection

***size*** The size of the data buffer in bytes

**Returns:**

Returns 1 if the ownership of the selection was successfully claimed, or 0 if unsuccessful.

#### 8.12.3.34 EAPI int ecore_x_selection_secondary_clear (void)

Release ownership of the secondary selection.

**Returns:**

Returns 1 if the selection was successfully cleared, or 0 if unsuccessful.

#### 8.12.3.35 EAPI int ecore_x_selection_secondary_set (Ecore_X_Window *w*, const void * *data*, int *size*)

Claim ownership of the SECONDARY selection and set its data.

**Parameters:**

***w*** The window to which this selection belongs

***data*** The data associated with the selection

***size*** The size of the data buffer in bytes

**Returns:**

Returns 1 if the ownership of the selection was successfully claimed, or 0 if unsuccessful.

#### 8.12.3.36 EAPI int ecore_x_selection_xdnd_clear (void)

Release ownership of the XDND selection.

**Returns:**

Returns 1 if the selection was successfully cleared, or 0 if unsuccessful.

#### 8.12.3.37 EAPI int ecore_x_selection_xdnd_set (Ecore_X_Window *w*, const void * *data*, int *size*)

Claim ownership of the XDND selection and set its data.

**Parameters:**

***w*** The window to which this selection belongs

***data*** The data associated with the selection

***size*** The size of the data buffer in bytes

**Returns:**

Returns 1 if the ownership of the selection was successfully claimed, or 0 if unsuccessful.

#### 8.12.3.38 EAPI void ecore_x_window_background_color_set (Ecore_X_Window *win*, unsigned short *r*, unsigned short *g*, unsigned short *b*)

Sets the background color of the given window.

**Parameters:**

***win*** The given window

***color*** The color to set to (i.e. 0xff0000)

#### 8.12.3.39 EAPI void ecore_x_window_cursor_show (Ecore_X_Window *win*, int *show*)

To be documented.

FIXME: To be fixed.

#### 8.12.3.40 EAPI void ecore_x_window_defaults_set (Ecore_X_Window *win*)

Sets the default properties for the given window.

The default properties set for the window are `WM_CLIENT_MACHINE` and `_NET_WM_PID`.

**Parameters:**

***win*** The given window.

#### 8.12.3.41 EAPI int ecore_x_window_depth_get (Ecore_X_Window *win*)

Retrieves the depth of the given window.

**Parameters:**

***win*** The given window.

**Returns:**

Depth of the window.

#### 8.12.3.42 EAPI void ecore_x_window_hide (Ecore_X_Window *win*)

Hides a window.

Synonymous to "unmapping" a window in X Window System terminology.

**Parameters:**

***win*** The window to hide.

#### 8.12.3.43 EAPI Ecore_X_Window* ecore_x_window_ignore_list (int * *num*)

Get the ignore list.

**Parameters:**

***num*** number of windows in the list

**Returns:**

list of windows to ignore

#### 8.12.3.44 EAPI void ecore_x_window_ignore_set (Ecore_X_Window *win*, int *ignore*)

Set if a window should be ignored.

**Parameters:**

***win*** The given window.

***ignore*** if to ignore

#### 8.12.3.45 EAPI Ecore_X_Atom ecore_x_window_prop_any_type (void)

To be documented.

FIXME: To be fixed.

#### 8.12.3.46 EAPI int ecore_x_window_prop_property_get (Ecore_X_Window *win*, Ecore_X_Atom *property*, Ecore_X_Atom *type*, int size __ *UNUSED* __, unsigned char ** *data*, int * *num*)

To be documented.

FIXME: To be fixed.

#### 8.12.3.47 EAPI void ecore_x_window_prop_property_set (Ecore_X_Window *win*, Ecore_X_Atom *property*, Ecore_X_Atom *type*, int *size*, void * *data*, int *number*)

To be documented.

FIXME: To be fixed.

#### 8.12.3.48 EAPI Ecore_X_WM_Protocol* ecore_x_window_prop_protocol_list_get (Ecore_X_Window *win*, int * *num_ret*)

To be documented.

FIXME: To be fixed.

#### 8.12.3.49 EAPI char* ecore_x_window_prop_string_get (Ecore_X_Window *win*, Ecore_X_Atom *type*)

Get a window string property.

**Parameters:**

***win*** The window

***type*** The property

Return window string property of a window. String must be free'd when done.

#### 8.12.3.50 EAPI void ecore_x_window_prop_string_set (Ecore_X_Window *win*, Ecore_X_Atom *type*, const char * *str*)

Set a window string property.

**Parameters:**

***win*** The window

***type*** The property

***str*** The string

Set a window string property

#### 8.12.3.51 EAPI Ecore_X_Window* ecore_x_window_root_list (int * *num_ret*)

Get a list of all the root windows on the server.

**Note:**

The returned array will need to be freed after use.

**Parameters:**

***num_ret*** Pointer to integer to put number of windows returned in.

**Returns:**

An array of all the root windows. `NULL` is returned if memory could not be allocated for the list, or if `num_ret` is `NULL`.

#### 8.12.3.52 EAPI void ecore_x_window_show (Ecore_X_Window *win*)

Shows a window.

Synonymous to "mapping" a window in X Window System terminology.

**Parameters:**

***win*** The window to show.

#### 8.12.3.53 EAPI int ecore_x_window_visible_get (Ecore_X_Window *win*)

Finds out whether the given window is currently visible.

**Parameters:**

***win*** The given window.

**Returns:**

1 if the window is visible, otherwise 0.

## 8.13 Ecore_X_Atoms.h File Reference

Ecore X atoms.

### 8.13.1 Detailed Description

Ecore X atoms.

## 8.14 Ecore_X_Cursor.h File Reference

Defines the various cursor types for the X Windows system.

### 8.14.1 Detailed Description

Defines the various cursor types for the X Windows system.

# Chapter 9

# Ecore Example Documentation

## 9.1 args_example.c

Shows how to set and retrieve the program arguments.

## 9.2 con_client_example.c

Shows how to write a simple client, that connects to the example server.

## 9.3 con_server_example.c

Shows how to write a simple server using the Ecore_Con library.

## 9.4 config_basic_example.c

Provides an example of how to use the basic configuration functions.See the file Ecore_Config.h for the full list of available functions.

## 9.5 config_listener_example.c

Shows how to set up a listener to listen for configuration changes.

## 9.6 event_handler_example.c

Shows how to use event handlers.

## 9.7 list_destroy_example.c

Shows how to set and use a destructor for an Ecore_List.

## 9.8 list_example.c

Provides a basic example of how to append to and traverse a list.

## 9.9 timer_example.c

Demonstrates use of the ecore_timer.

## 9.10 x_window_example.c

Shows the basics of using the X Windows system through Ecore functions.

# Chapter 10

# Ecore Page Documentation

## 10.1 Todo List

**page Ecore** (1.0) Document API

**Global ecore_fb_callback_gain_set** Documentation: Find out what this does.

**Global ecore_fb_callback_lose_set** Documentation: Find out what this does.

**Group Ecore_Path_Group** Give this a better description.

**Global ecore_con_server_connected_get** Check that this function does what the documenter believes it does.

**Global ecore_fb_led_blink_set** Documentation: Work out what speed the units are in.

**Global ecore_ipc_server_add** Need to add protocol type parameter to this function.

**Global ecore_ipc_server_connect** Need to add protocol type parameter.

**Global ecore_ipc_server_send** This function needs to become an IPC message.

**Global ecore_ipc_server_send** Fix up the documentation: Make sure what ref_to and response are.

**Global ecore_ipc_client_send** This function needs to become an IPC message.

**Global ecore_ipc_client_send** Make sure ref_to and response parameters are described correctly.

**Page Ecore Abstract Data Types** Finish this.

## 10.2 The Ecore Main Loop

### 10.2.1 What is Ecore?

Ecore is a clean and tiny event loop library with many modules to do lots of convenient things for a programmer, to save time and effort.

It's small and lean, designed to work on embedded systems all the way to large and powerful multi-cpu workstations. It serialises all system signals, events etc. into a single event queue, that is easily processed without needing to worry about concurrency. A properly written, event-driven program using this kind of programming doesn't need threads, nor has to worry about concurrency. It turns a program into a state machine, and makes it very robust and easy to follow.

Ecore gives you other handy primitives, such as timers to tick over for you and call specified functions at particular times so the programmer can use this to do things, like animate, or time out on connections or tasks that take too long etc.

Idle handlers are provided too, as well as calls on entering an idle state (often a very good time to update the state of the program). All events that enter the system are passed to specific callback functions that the program sets up to handle those events. Handling them is simple and other Ecore modules produce more events on the queue, coming from other sources such as file descriptors etc.

Ecore also lets you have functions called when file descriptors become active for reading or writing, allowing for streamlined, non-blocking IO.

Here is an exmaple of a simple program and its basic event loop flow:

### 10.2.2 How does Ecore work?

Ecore is very easy to learn and use. All the function calls are designed to be easy to remember, explicit in describing what they do, and heavily name-spaced. Ecore programs can start and be very simple.

For example:

```
#include <Ecore.h>

int main(int argc, const char **argv)
{
  ecore_init();
  ecore_app_args_set(argc, argv);
  ecore_main_loop_begin();
  ecore_shutdown();
  return 0;
}
```

This program is very simple and does't check for errors, but it does start up and begin a main loop waiting for events or timers to tick off. This program doesn't set up any, but now we can expand on this simple program a little more by adding some event handlers and timers.

```
#include <Ecore.h>

Ecore_Timer         *timer1     = NULL;
Ecore_Event_Handler *handler1   = NULL;
double               start_time = 0.0;

int timer_func(void *data)
{
  printf("Tick timer. Sec: %3.2f\n", ecore_time_get() - start_time);
  return 1;
}

int exit_func(void *data, int ev_type, void *ev)
{
  Ecore_Event_Signal_Exit *e;

  e = (Ecore_Event_Signal_Exit *)ev;
  if (e->interrupt)      printf("Exit: interrupt\n");
  else if (e->quit)      printf("Exit: quit\n");
  else if (e->terminate) printf("Exit: terminate\n");
  ecore_main_loop_quit();
  return 1;
}

int main(int argc, const char **argv)
{
  ecore_init();
  ecore_app_args_set(argc, argv);
  start_time = ecore_time_get();
  handler1 = ecore_event_handler_add(ECORE_EVENT_SIGNAL_EXIT, exit_func, NULL);
  timer1 = ecore_timer_add(0.5, timer_func, NULL);
  ecore_main_loop_begin();
  ecore_shutdown();
  return 0;
}
```

In the previous example, we initialize our application and get the time at which our program has started so we can calculate an offset. We set up a timer to tick off in 0.5 seconds, and since it returns 1, will keep ticking off every 0.5 seconds until it returns 0, or is deleted by hand. An event handler is set up to call a function - exit_func(), whenever an event of type ECORE_EVENT_SIGNAL_EXIT is received (CTRL-C on the command line will cause such an event to happen). If this event occurs it tells you what kind of exit signal was received, and asks the main loop to quit when it is finished by calling ecore_main_loop_quit().

The handles returned by ecore_timer_add() and ecore_event_handler_add() are only stored here as an example. If you don't need to address the timer or event handler again you don't need to store the result, so just call the function, and don't assign the result to any variable.

This program looks slightly more complex than needed to do these simple things, but in principle, programs don't get any more complex. You add more event handlers, for more events, will have more timers and such, BUT it all follows the same principles as shown in this example.

## 10.3 The Enlightened Property Library

The Enlightened Property Library (Ecore_Config) is an adbstraction from the complexities of writing your own configuration.

It provides many features using the Enlightenment 17 development libraries.

To use the library, you:

- Set the default values of your properties.
- Load the configuration from a file. You must set the default values first, so that the library knows the correct type of each argument.

The following examples show how to use the Enlightened Property Library:

- config_basic_example.c
- config_listener_example.c

## 10.4 Ecore Abstract Data Types

This page briefly describes the different abstract data types that are provided by the Ecore library for general usage.

You need to include the Ecore_Data.h to use them.

### 10.4.1 List

A list is a simple data type where one each piece of data points to another piece of data.

Associated modules that describe the List ADT include:

- List Creation/Destruction Functions
- List Item Adding Functions
- List Item Removing Functions
- List Traversal Functions
- List Node Functions

Examples involving lists include:

- list_example.c

### 10.4.2 Doubly Linked List

A doubly linked list is like a linked list, only each piece of data can also point to the piece before it. In other words, you can traverse a doubly linked list in both directions.

Associated modules that describe the DList ADT include:

- Doubly Linked List Creation/Destruction Functions
- Doubly Linked List Adding Functions
- Doubly Linked List Removing Functions

### 10.4.3 Hash

A hash is an abstract data type where one value is associated with another value. Instead of each element of the group being accessible using a number, each element is accessed using another object.

Associated modules that describe the Hash ADT include:

- Hash Creation Functions
- Hash Destruction Functions
- Hash Data Functions

**Todo**

Finish this.

## 10.5 X Window System

The Ecore library includes a wrapper for handling the X window system.

This page briefly explains what the X window system is and various terms that are used.

# Index