# FOSA Reference Manual

D-EP3v1

Generated by Doxygen 1.4.6

Mon Jun 2 13:52:35 2008

# Contents



# 1 FOSA Module Index

## 1.1 FOSA Modules

Here is a list of all modules:



# 2 FOSA Hierarchical Index

## 2.1 FOSA Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:



# 3 FOSA Data Structure Index

## 3.1 FOSA Data Structures

Here are the data structures with brief descriptions:

# 4 FOSA Module Documentation

## 4.1 FOSA Private Interfaces

### Modules

- Application Defined Scheduling
- Clocks and Timers
- Long Jumps
- Mutexes and Condvars
- Thread and Signals

### 4.1.1 Detailed Description

FOSA is an OS adaption layer that encapsulates all POSIX types and functions into neutral names so that FRSH can compile and be used in non-POSIX operating systems such as OSE.

It is divided in two parts:

- FRSH_FOSA: Types visible to the application via FRSH_API and the functions to manage them (thread, signals).
- FOSA: Types and functions only used within FRSH.

The former reside in the FRSH subversion directory and the latter have their own. They need to be separated because the application must not see FOSA itself.

For simplicity, we have chosen to hide the operation function on signals and mutexes with the assumption that a direct mapping exists for frsh_signal_t, frsh_signal_info_t and frsh_mutext_t in the native OS.

Since there are some parts which are platform dependent a define has been introduced for each platform. Currently the supported defines are:

-DRT_LINUX -DPARTIKLE -DOSE -DMARTE_OS -DAQuoSA

This module contains all other modules that are internal to the FRSH implementation.

Note that to compile FOSA objects an include path towards FRSH is needed: -I<frsh_include_directory>.

## 4.2 Application Defined Scheduling

### Data Structures

- struct fosa_ads_scheduler_ops_t

### Typedefs

- typedef FOSA_ADS_ACTIONS_T_OPAQUE fosa_ads_actions_t
- typedef int fosa_ads_urgency_t

### Enumerations

- enum fosa_ads_error_cause_t { **FOSA_ADS_THREAD_NOT_ATTACHED**, **FOSA_ADS_-INVALID_ACTION** }

### Functions

- int fosa_ads_scheduler_create (const fosa_ads_scheduler_ops_t *scheduler_ops, size_t scheduler_-data_size, void *init_args, size_t init_args_size)
- int fosa_thread_attr_set_appscheduled (fosa_thread_attr_t *attr, bool appscheduled)
- int fosa_thread_attr_get_appscheduled (const fosa_thread_attr_t *attr, bool *appscheduled)
- int fosa_thread_attr_set_appsched_params (fosa_thread_attr_t *attr, const void *param, size_-t paramsize)
- int fosa_thread_attr_get_appsched_params (const fosa_thread_attr_t *attr, void *param, size_-t *paramsize)
- int fosa_ads_set_appscheduled (fosa_thread_id_t thread, bool appscheduled)
- int fosa_ads_get_appscheduled (fosa_thread_id_t thread, bool *appscheduled)
- int fosa_ads_set_appsched_params (fosa_thread_id_t thread, const void *param, size_t paramsize)
- int fosa_ads_get_appsched_params (fosa_thread_id_t thread, void *param, size_t *paramsize)
- int fosa_adsactions_add_reject (fosa_ads_actions_t *sched_actions, fosa_thread_id_t thread)
- int fosa_adsactions_add_activate (fosa_ads_actions_t *sched_actions, fosa_thread_id_t thread, fosa_ads_urgency_t urgency)
- int fosa_adsactions_add_suspend (fosa_ads_actions_t *sched_actions, fosa_thread_id_t thread)
- int fosa_adsactions_add_timeout (fosa_ads_actions_t *sched_actions, fosa_clock_id_t clock_id, const fosa_abs_time_t *at_time)
- int fosa_adsactions_add_thread_notification (fosa_ads_actions_t *sched_actions, fosa_thread_id_t thread, fosa_clock_id_t clock_id, const fosa_abs_time_t *at_time)
- int fosa_ads_set_handled_signal_set (fosa_signal_t set[ ], int size)
- int fosa_signal_queue_scheduler (fosa_signal_t signal, fosa_signal_info_t info)
- int fosa_ads_invoke_withdata (const void *msg, size_t msg_size, void *reply, size_t *reply_size)

#### 4.2.1 Detailed Description

This module defines the function and types for an abstraction of the Application Defined Scheduling.

#### 4.2.2 Typedef Documentation

##### 4.2.2.1 typedef FOSA_ADS_ACTIONS_T_OPAQUE fosa_ads_actions_t

ADS actions

This type is used to represent a list of scheduling actions that the scheduler will later request to be executed by the system. The pos- sible actions are of the following kinds:

- reject a thread that has requested attachment to this scheduler
- activate an application-scheduled thread with the desired value of urgency
- suspend an application-scheduled thread
- program a timeout
- program a timed notification associated to a particular application-scheduled thread.

No comparison or assignment operators are defined for this type

#### 4.2.2.2 typedef int fosa_ads_urgency_t

The urgency used to orde the threads of the same priority in the underlying scheduler. Support for urgency scheduling is required for supporting the hierarchhical scheduling module

### 4.2.3 Enumeration Type Documentation

#### 4.2.3.1 enum fosa_ads_error_cause_t

Causes of error in the appsched_error primitive operation

### 4.2.4 Function Documentation

#### 4.2.4.1 int fosa_ads_scheduler_create (const fosa_ads_scheduler_ops_t * *scheduler_ops*, size_t *scheduler_data_size*, void * *init_args*, size_t *init_args_size*)

fosa_ads_scheduler_create()

Create the application defined scheduler

The application defined scheduler is created with the primitive operations specified in the object pointed to by scheduler_ops.

The clock used to read the time immediately before the invocation of each primitive operation, to be reported to the scheduler via the current_time parameter of each primitive operation is the FOSA_CLOCK_REALTIME clock.

The scheduler_data_size parameter is used to request that a memory area of this size must be created and reserved for the scheduler to store its state. A pointer to this area is passed to the scheduler operations in the sched_data parameter.

Parameter init_arg points to an area that contains configuration information for the scheduler. The function creates a memory area of init_arg_size bytes and copies into it the area pointed by arg. A pointer to this new created area will be passed to the primitive operation init() in its arg parameter.

This function must be called before any other function in this header file.

In addition it must be called at a priority level no greater than the priority at which the scheduler operations execute. This priority is defined as the maximum SCHED_FIFO priority in the system minus the configuration parameter FOSA_ADS_SCHEDULER_PRIO_DIFF.

Returns 0 if successful; otherwise it returns an error code:

FOSA_EINVAL: The value of scheduler_ops was invalid

FOSA_EAGAIN: The system lacks enough resources to create the scheduler

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.2.4.2 int fosa_thread_attr_set_appscheduled (fosa_thread_attr_t * *attr*, bool *appscheduled*)

fosa_thread_attr_set_appscheduled()

Set the appscheduled attribute of a thread attributes object

This function is used to set the appscheduled attribute in the object pointed to by attr. This attribute controls the kind of scheduling used for threads created with it. If true, the thread is scheduled by the application scheduler. If not, it is scheduled by the system under a fixed priority scheduler

Returns 0 if successful; otherwise it returns an error code:

FOSA_EINVAL: The value of attr is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.2.4.3 int fosa_thread_attr_get_appscheduled (const fosa_thread_attr_t * *attr*, bool * *appscheduled*)

fosa_thread_attr_get_appscheduled()

Get the appscheduled attribute of a thread attributes object

This function is used to get the appscheduled attribute in the object pointed to by attr. This attribute controls the kind of scheduling used for threads created with it. If true, the thread is scheduled by the application scheduler. If not, it is scheduled by the system under a fixed priority scheduler.

Returns 0 if successful; otherwise it returns an error code:

FOSA_EINVAL: The value of attr is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.2.4.4 int fosa_thread_attr_set_appsched_params (fosa_thread_attr_t * *attr*, const void * *param*, size_t *paramsize*)

fosa_thread_attr_set_appsched_params()

Set the appsched_param attribute of a thread attributes object

This function is used to set the appsched_param attribute in the object pointed to by attr. For those threads with appscheduled set to true, this attribute represents the application-specific scheduling parameters. If successful, the function shall set the size of the appsched_param attribute to the value specified by paramsize, and shall copy the scheduling parameters occupying paramsize bytes and pointed to by param into that attribute

Returns 0 if successful; otherwise it returns an error code:

FOSA_EINVAL: The value of attr is invalid, or paramsize is less than zero or larger than FOSA_ADS_-SCHEDPARAM_MAX

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.2.4.5 int fosa_thread_attr_get_appsched_params (const fosa_thread_attr_t * *attr*, void * *param*, size_t * *paramsize*)

fosa_thread_attr_get_appsched_params()

Get the appsched_param attribute of a thread attributes object

This function is used to get the appsched_param attribute from the object pointed to by attr. For those threads with appscheduled set to true, this attribute represents the application-specific scheduling parameters. If successful, the function shall set the value pointed to by paramsize to the size of the appsched_param attribute, and shall copy the scheduling parameters occupying paramsize bytes into the variable pointed to by param. This variable should be capable of storing a number of bytes equal to paramsize.

Returns 0 if successful; otherwise it returns an error code:

FOSA_EINVAL: The value of attr is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.2.4.6 int fosa_ads_set_appscheduled (fosa_thread_id_t *thread*, bool *appscheduled*)

fosa_ads_set_appscheduled()

Dynamically set the appscheduled attribute of a thread

This function is used to dynamically set the appscheduled attribute of the thread identified by thread. This attribute controls the kind of scheduling used for threads created with it. If true, the thread is scheduled by the application scheduler. If not, it is scheduled by the system under a fixed priority scheduler.

Returns 0 if successful; otherwise it returns an error code:

FOSA_EINVAL: The value of thread is invalid

FOSA_EREJECT: the attachment of the thread to the frsh schehduler was rejected by the frsh scheduler possibly because of incorrect attributes, or because the requested minimum capacity cannot be guaranteed

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.2.4.7 int fosa_ads_get_appscheduled (fosa_thread_id_t *thread*, bool * *appscheduled*)

fosa_ads_getappscheduled()

Dynamically get the appscheduled attribute of a thread

This function is used to dynamically get the appscheduled attribute of the thread identified by thread. This attribute controls the kind of scheduling used for threads created with it. If true, the thread is scheduled by the application scheduler. If not, it is scheduled by the system under a fixed priority scheduler

Returns 0 if successful; otherwise it returns an error code:

FOSA_EINVAL: The value of thread is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.2.4.8 int fosa_ads_set_appsched_params (fosa_thread_id_t *thread*, const void * *param*, size_t *paramsize*)

fosa_ads_setappschedparam()

Dynamically set the appsched_param attribute of a thread

This function is used to dynamically set the appsched_param attribute of the thread identified by thread. For those threads with appscheduled set to true, this attribute represents the application-specific scheduling parameters. If successful, the function shall set the size of the appsched_param attribute to the value specified by paramsize, and shall copy the scheduling parameters occupying paramsize bytes and pointed to by param into that attribute

Returns 0 if successful; otherwise it returns an error code:

FOSA_EINVAL: The value of thread is invalid, or paramsize is less than zero or larger than FOSA_ADS_-SCHEDPARAM_MAX

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.2.4.9 int fosa_ads_get_appsched_params (fosa_thread_id_t *thread*, void * *param*, size_t * *paramsize*)

fosa_ads_get_appsched_params()

Dynamically get the appsched_param attribute of a thread

This function is used to dynamically get the appsched_param attribute of the thread identified by thread. For those threads with appscheduled set to true, this attribute represents the application-specific scheduling parameters. If successful, the function shall set the variable pointed to by paramsize to the size of the appsched_param attribute, and shall copy the scheduling parameters occupying paramsize bytes into the variable pointed to by param. This variable should be capable of storing a number of bytes equal to paramsize.

Returns 0 if successful; otherwise it returns an error code:

FOSA_EINVAL: The value of thread is invalid, or paramsize is less than zero or larger than FOSA_ADS_-SCHEDPARAM_MAX

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications.

#### 4.2.4.10 int fosa_adsactions_add_reject (fosa_ads_actions_t * *sched_actions*, fosa_thread_id_t *thread*)

fosa_adsactions_add_reject()

Add a reject-thread action

This function adds a thread-reject action to the object referenced by sched_actions, that will serve to notify that the thread identified by thread has not been accepted by the scheduler to be scheduled by it, possibly because the thread contained invalid application scheduling attributes, or because there are not enough resources for the new thread. At the end of the new_thread() scheduler primitive operation, the parent of the rejected thread waiting on a fosa_thread_create() or the rejected thread itself waiting on a fosa_ads_set_-appscheduled() function shall complete the function with an error code of FOSA_EREJECT. If no reject-thread action is added during the new_thread() scheduler primitive operation, the thread is accepted to be scheduled by the scheduler and the associated fosa_thread_create() or the fosa_ads_set_appscheduled() function shall be completed without error. For the function to succeed, it has to be called from the new_-thread() primitive operation and for the thread that is requesting attachment to the scheduler.

Returns 0 if successful; otherwise it returns an error code:

FOSA_ENOMEM: There is insufficient memory to add this action

FOSA_EPOLICY: The thread specified by thread is not the one requesting attachment to the scheduler, or the function is not being called from the new_thread primitive operation

FOSA_EINVAL: The value specified by sched_actions is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.2.4.11 int fosa_adsactions_add_activate (fosa_ads_actions_t * *sched_actions*, fosa_thread_id_t *thread*, fosa_ads_urgency_t *urgency*)

fosa_adsactions_add_activate()

Add a thread-activate action

This function adds a thread-activate action to the object referenced by sched_actions. In case the thread had been previously suspended via posix_appsched_actions_addsuspend(), it will be activated at the end

of the primitive operation.

In those implementations that do not support urgency scheduling, the urgencu value is ignored. These implementations cannot support the frsh hierarchical scheduling module.

In those implementations supporting urgency-scheduling, the action will cause the change of the urgency of the thread to the value specified in the urgency argument. Besides, if the thread was already active at the time the thread-activate action is executed, the change or urgency will imply a reordering of the thread in its priority queue, so that for threads of the same priority, those with more urgency will be scheduled before those of less urgency.

Returns 0 if successful; otherwise it returns an error code:

FOSA_ENOMEM: There is insufficient memory to add this action

FOSA_EPOLICY: The thread specified by thread has its appscheduled attribute set to false

FOSA_EINVAL: The value specified by sched_actions is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.2.4.12 int fosa_adsactions_add_suspend (fosa_ads_actions_t * *sched_actions*, fosa_thread_id_t *thread*)

fosa_adsactions_add_suspend()

Add a thread-suspend action

This function adds a thread-suspend action to the object referenced by sched_actions, that will cause the thread identified by thread to be suspended waiting for a thread-activate action at the end of the scheduler operation. If the thread was already waiting for a thread-activate action the thread-suspend action has no effect. It is an error trying to suspend a thread that is blocked by the operating system.

Returns 0 if successful; otherwise it returns an error code:

FOSA_ENOMEM: There is insufficient memory to add this action

FOSA_EPOLICY: The thread specified by thread has its appscheduled attribute set to false

FOSA_EINVAL: The value specified by sched_actions is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.2.4.13 int fosa_adsactions_add_timeout (fosa_ads_actions_t * *sched_actions*, fosa_clock_id_t *clock_id*, const fosa_abs_time_t * *at_time*)

fosa_adsactions_add_timeout()

Add a timeout action

This function adds a timeout action to the object referenced by sched_actions, that will cause the timeout() scheduler operation to be invoked if no other scheduler operation is invoked before timeout expires. The timeout shall expire when the clock specified by clock_id reaches the absolute time specified by the at_time argument.

Returns 0 if successful; otherwise it returns an error code:

FOSA_ENOMEM: There is insufficient memory to add this action

FOSA_EINVAL: The value specified by sched_actions is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then

terminate the FRSH implementation and dependant applications

#### 4.2.4.14 int fosa_adsactions_add_thread_notification (fosa_ads_actions_t * *sched_actions*, fosa_thread_id_t *thread*, fosa_clock_id_t *clock_id*, const fosa_abs_time_t * *at_time*)

fosa_adsactions_add_thread_notification()

Add a timed-thread-notification action

This function adds a thread-notification action associated with the thread specified in the thread argument that will cause the notification_for_thread() scheduler operation to be invoked at the time specified by at_time. This operation shall be invoked when the clock specified by clock_id reaches the absolute time specified by the at_time argument. In particular, a cpu-time clock may be used for parameter clock_id.Only one thread-notification can be active for each thread and clock. Calling the function shall remove the former thread-notification, if any, that had been programmed for the same thread and clock. A value of NULL for parameter at_time is used to cancel a previous thread-notification, if any, for the thread specified by thread and the clock specified by clock_id.

Returns 0 if successful; otherwise it returns an error code:

FOSA_ENOMEM: There is insufficient memory to add this action

FOSA_EPOLICY: The thread specified by thread has its appscheduled attribute set to false

FOSA_EINVAL: The value specified by sched_actions is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.2.4.15 int fosa_ads_set_handled_signal_set (fosa_signal_t *set*[ ], int *size*)

fosa_ads_set_handled_signal_set()

Specifiy the set of signals that will be handled by the application scheduler

This function is used to dynamically set the set of signals that are handled by the application scheduler. When a signal included in this set is generated, the signal() primitive operation of the application scheduler shall be executed. When a signal in tis set is generated, it shall always imply the execution of the signal() primitive operation, regardless of whether that signal could be accepted by some other thread. Once the signal() primitive operation is executed the signal is consumed, so no signal handlers shall be executed and no threads using a sigwait operation shall return for that particular signal instance. For this function to succeed, it has to be called from a primitive operation of a scheduler.

The size of the array is specified by argument size.

Returns 0 if successful; otherwise it returns an error code:

FOSA_EPOLICY: The function has not been called from a scheduler primitive operation

FOSA_EINVAL: The value specified by set is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.2.4.16 int fosa_signal_queue_scheduler (fosa_signal_t *signal*, fosa_signal_info_t *info*)

fosa_signal_queue_scheduler()

Queue a signal destinated to the scheduler

This is a special case of fosa_signal_queue() in which the destinator is the scheduler itself. It is needed by the service thread to notify the results to the scheduler.

The problem with this case is that, depending on the implementation, this call would be translated to a true signal or to a scheduler notification message.

Besides for the scheduler we don't have always a destinator thread_id needed in fosa_signal_queue for OSE.

So the fosa implementation will solve this issue internally.

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the signal specified by signal is not between FOSA_SIGNAL_MIN and FOSA_SIGNAL_MAX

FOSA_EAGAIN: no resources are available to queue the signal; the maximum number of queued signals has been reached, or a systemwide resource limit has been exceeded

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.2.4.17 int fosa_ads_invoke_withdata (const void * *msg*, size_t *msg_size*, void * *reply*, size_t * *reply_size*)

fosa_ads_invoke_withdata()

Explicitly invoke the scheduler, with data

This function can be used by any thread in the process to invoke the ads scheduler or to share data with it.

If successful, the function shall cause the execution of the primitive operation explicit_call_with_data() of the ads scheduler with its thread parameter equal to the thread ID of the calling thread, and its msg_size parameter equal to msg_size. In addition, if msg_size is larger than zero, the function shall make available to the scheduler a memory area whose contents are identical to the memory area pointed to by msg in the msg parameter of the explicit_call_with_data() primitive operation (note that copying the information is not needed).

The function shall not return until the system has finished execution of the explicit_call_with_data() primitive operation. If the reply argument is non NULL, the memory area pointed to by the reply parameter of explicit_call_with_data() primitive operation is copied into the memory area pointed to by reply, and its size is copied into the variable pointed to by reply_size. The size of the reply information is limited to the value FOSA_ADS_SCHEDINFO_MAX.

The function shall fail if the size specified by msg_size is larger than FOSA_ADS_SCHEDINFO_MAX. The function shall fail if primitive operation explicit_call_with_data() is set to NULL for the ads scheduler.

Returns 0 if successful; otherwise it returns an error code:

FOSA_EPOLICY: The function been called from inside a scheduler primitive operation

FOSA_EINVAL: The value of msg_size is less than zero or larger than FOSA_ADS_SCHEDINFO_MAX

FOSA_EMASKED: The operation cannot be executed because the primitive operation explicit_call_with_data() is set to NULL

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

## 4.3 Clocks and Timers

### Defines

- #define **FOSA_SYSTEM_CLOCK** FOSA_SYSTEM_CLOCK_OPAQUE

### Typedefs

- typedef FOSA_REL_TIME_T_OPAQUE **fosa_rel_time_t**
- typedef FOSA_ABS_TIME_T_OPAQUE **fosa_abs_time_t**
- typedef FOSA_CLOCK_ID_T_OPAQUE **fosa_clock_id_t**
- typedef FOSA_TIMER_ID_T_OPAQUE **fosa_timer_id_t**

### Functions

- int fosa_clock_get_time (fosa_clock_id_t clockid, fosa_abs_time_t *current_time)
- int fosa_thread_get_cputime_clock (fosa_thread_id_t tid, fosa_clock_id_t *clockid)
- int fosa_timer_create (fosa_clock_id_t clockid, fosa_signal_t signal, fosa_signal_info_t info, fosa_timer_id_t *timerid)
- int fosa_timer_create_with_receiver (fosa_clock_id_t clockid, fosa_signal_t signal, fosa_signal_info_t info, fosa_timer_id_t *timerid, fosa_thread_id_t receiver)
- int fosa_timer_delete (fosa_timer_id_t timerid)
- int fosa_rel_timer_arm (fosa_timer_id_t timerid, const fosa_rel_time_t *value)
- int fosa_abs_timer_arm (fosa_timer_id_t timerid, const fosa_abs_time_t *value)
- int fosa_timer_get_remaining_time (fosa_timer_id_t timerid, fosa_rel_time_t *remaining_time)
- int fosa_timer_disarm (fosa_timer_id_t timerid, fosa_rel_time_t *remaining_time)

#### 4.3.1 Detailed Description

This module defines the types and functions to abstract clocks and timers for the FRSH implementation.

#### 4.3.2 Function Documentation

##### 4.3.2.1 int fosa_clock_get_time (fosa_clock_id_t *clockid*, fosa_abs_time_t * *current_time*)

fosa_get_time()

Get the time from a clock

This function sets the variable pointed to by current_time to the current value of the clock specified by clockid, which may be the FOSA_CLOCK_REALTIME constant or a value obtained with fosa_get_cputime_clock()

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the value of clockid is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

##### 4.3.2.2 int fosa_thread_get_cputime_clock (fosa_thread_id_t *tid*, fosa_clock_id_t * *clockid*)

fosa_get_cputime_clock()

Get the identifier of a cpu-time clock

This function stores in the variable pointed to by clockid the identifier of a cpu-time clock for the thread specified by tid.

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the value of tid is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.3.2.3 int fosa_timer_create (fosa_clock_id_t *clockid*, fosa_signal_t *signal*, fosa_signal_info_t *info*, fosa_timer_id_t * *timerid*)

fosa_timer_create()

Create a one-shot timer

This function creates a timer based on the clock specified by clock, and associates to this timer a notification mechanism consisting of a signal and associated information. Initially, the timer is in the disarmed state, i.e., not counting time. It can be armed to start counting time with fosa_timer_arm().

The function stores the identifier of the newly created timer in the variable pointed to by timerid.

When the timer expires, the signal number specified by signal will be sent together with the information specified by info, to the thread that armed the timer (

**See also:**
fosa_timer_arm()).

In those implementations that do not support queueing a signal with information to a thread (such as POSIX), the signal may be sent to any thread that is waiting for this signal via fosa_signal_wait(). Portability can be ensured by having the receiver thread be the one who is waiting for the signal.

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the value of clockid or signal is invalid

FOSA_EAGAIN: the system lacks enough resources to create the timer

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.3.2.4 int fosa_timer_create_with_receiver (fosa_clock_id_t *clockid*, fosa_signal_t *signal*, fosa_signal_info_t *info*, fosa_timer_id_t * *timerid*, fosa_thread_id_t *receiver*)

fosa_timer_create_with_receiver()

Create a one-shot timer with a specific signal receiver thread

This function creates a timer in the same way as fosa_timer_create, except that the signal generated when the timer expires is sent to the thread specified by receiver

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the value of clockid or signal is invalid

FOSA_EAGAIN: the system lacks enough resources to create the timer

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.3.2.5 int fosa_timer_delete (fosa_timer_id_t *timerid*)

fosa_timer_delete()

Delete a timer

The function deletes the timer specified by timerid, which becomes unusable. If the timer was armed, it is automatically disarmed before deletion.

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the value of timerid is not valid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.3.2.6 int fosa_rel_timer_arm (fosa_timer_id_t *timerid*, const fosa_rel_time_t * *value*)

fosa_rel_timer_arm()

Arm a timer with a relative time interval

The timer specified by timer is armed and starts counting time.

The value pointed to by value is the relative interval that must elapse for the timer to expire. Negative values cause the timer to expire immediately.

The time is measured with the clock associated with the timer when it was created.

If the timer was already armed, the previous time or interval is discarded and the timer is rearmed with the new value.

When the timer expires, it is disarmed.

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the value of timerid or value is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.3.2.7 int fosa_abs_timer_arm (fosa_timer_id_t *timerid*, const fosa_abs_time_t * *value*)

fosa_abs_timer_arm()

Arm a timer that will expire in an absolute time instant.

The timer specified by timer is armed and starts counting time.

The value pointed to by value is the absolute time at which the timer will expire. If value specifies a time instant in the past, the timer expires immediately.

The time is measured with the clock associated with the timer when it was created.

If the timer was already armed, the previous time or interval is discarded and the timer is rearmed with the new value.

When the timer expires, it is disarmed.

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the value of timerid or value is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.3.2.8 int fosa_timer_get_remaining_time (fosa_timer_id_t *timerid*, fosa_rel_time_t * *remaining_time*)

fosa_timer_get_remaining_time()

Get the remaining time for timer expiration

Returns the relative remaining time for timer expiration. If the clock is a CPU clock it returns the time as if the thread was executing constantly.

If the timer is disarmed it returns 0.

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the value of timerid or value is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.3.2.9 int fosa_timer_disarm (fosa_timer_id_t *timerid*, fosa_rel_time_t * *remaining_time*)

fosa_timer_disarm()

Disarm a timer and optionally obtain remaining time before expiration

The timer specified by timer is disarmed, and will not expire unless it is rearmed. If the timer was already disramed, the function has no effect.

If the pointer remaining_time is != NULL, the remaining time before expiration will be returned in that pointer. If the timer was disarmed a 0 value will be set.

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the value of timerid or value is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

## 4.4 Long Jumps

### Functions

- int fosa_long_jump_save_context (fosa_long_jump_context_t *context)
- int fosa_long_jump_was_performed (const fosa_long_jump_context_t *context, int *jumped)
- int fosa_long_jump_install_handler (fosa_signal_t *signal, fosa_thread_id_t *handler)

### 4.4.1 Detailed Description

This module defines the types and functions that allow te application to abort a piece of running code; it can be used to stop an action that is overrunning its budget and needs to be aborted, without aborting the whole thread on which it is based.

The model is that the application installs a long-jump handler for each thread that potentially needs to be aborted through the long jump mechanism. As a result of installing the handler it gets back a signal identifier. This signal is later used to notify the handler that the thread needs to be aborted. Previous to sending the signal, the application must store the context of the thread so that when it is aborted the saved context can be recovered. The signal may be sent to the handler from any thread, but must contain as attached information a pointer to the variable where the context was saved.

The implementation may internally choose to implement a single long jump handler for all threads (and use a single signal), or one for each thread (requiring the reservation of a pool of signals).

### 4.4.2 Function Documentation

#### 4.4.2.1 int fosa_long_jump_save_context (fosa_long_jump_context_t * *context*)

fosa_long_jump_save_context

Save the context of the current thread for a future long jump

This function stores in context the information required to modify the stack of the calling thread so that a later long jump may be executed in the future to restore this context

This function stores in 'context' the thread id, the registers, and the stack frame of the calling thread. This information can be used by the long jump handler to change the stack of the task so that when it is scheduled again it returns to the end of this function

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the value of context is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.4.2.2 int fosa_long_jump_was_performed (const fosa_long_jump_context_t * *context*, int * *jumped*)

fosa_long_jump_was_performed

Check whether the current thread suffered a long jump or not

This function should be invoked after fosa_long_jump_save_context to determine whether the current thread is executing normally, or has suffered a long jump to the point where the context was saved. If invoked after a direct invocation to fosa_long_jump_save_context, the function shall set the variable pointed to by jumped to zero. If invoked after returning from fosa_long_jump_save_context due to a long jump caused by the long jump handler, the function shall set this variable to 1.

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the value of context is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.4.2.3 int fosa_long_jump_install_handler (fosa_signal_t * *signal*, fosa_thread_id_t * *handler*)

fosa_long_jump_install_handler

Install a long jump handler for the calling thread

This function shall install a handler that is capable of causing a long jump operation that restores the context of a thread to a previously saved value. If the handler has already been installed for this thread the previously installed handler will be used and the call shall succeed.

The long-jump handler is waiting for a signal to notify that a thread context should be restored. This signal must carry attached to it a pointer to the variable where the thread context was saved. The thread referenced in that context will have its internal context restored to the point where it was saved. For this restore operation to work properly, the program frame where the thread saved its context must still be valid.

Depending on the implementation a given thread may also install a signal handler or a signal handler thread that is capable of executing the actions required to restore the context of a thread from the appropriate context. For instance, in POSIX it is necesaray that the context is restored from the same thread being restored, usually from a signal handler of that thread.

The function shall store in the variable pointed to by signal the identifier of the signal that must be used to notify the request for a long jump to be executed. In the variable pointed to by handler, it shall store the thread id to which the signal must be sent. The signal must be sent with its attached information set to a pointer to the variable of type fosa_long_jump_context_t where the context of the thread to be restored was saved.

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the value of context is invalid FOSA_ENOMEM: there are no resources to satisfy the call at this time

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

## 4.5 Mutexes and Condvars

### Functions

- int fosa_mutex_init (fosa_mutex_t *mutex, int prioceiling)

- int fosa_mutex_destroy (fosa_mutex_t ∗mutex)
- int fosa_mutex_set_prioceiling (fosa_mutex_t ∗mutex, int new_ceiling, int ∗old_ceiling)
- int fosa_mutex_get_prioceiling (const fosa_mutex_t ∗mutex, int ∗ceiling)
- int fosa_mutex_lock (fosa_mutex_t ∗mutex)
- int fosa_mutex_trylock (fosa_mutex_t ∗mutex)
- int fosa_mutex_unlock (fosa_mutex_t ∗mutex)
- int fosa_cond_init (fosa_cond_t ∗cond)
- int fosa_cond_destroy (fosa_cond_t ∗cond)
- int fosa_cond_signal (fosa_cond_t ∗cond)
- int fosa_cond_broadcast (fosa_cond_t ∗cond)
- int fosa_cond_wait (fosa_cond_t ∗cond, fosa_mutex_t ∗mutex)
- int fosa_cond_timedwait (fosa_cond_t ∗cond, fosa_mutex_t ∗mutex, const fosa_abs_time_t ∗abstime)

### 4.5.1 Detailed Description

This module defines the types and functions to abstract mutexes and conditional variables for the FRSH implementation.

### 4.5.2 Function Documentation

#### 4.5.2.1 int fosa_mutex_init (fosa_mutex_t ∗ *mutex*, int *prioceiling*)

fosa_mutex_init()

Initialize a frsh mutex

The mutex pointed to by mutex is initialized as a mutex using the priority ceiling protocol. A priority ceiling of prioceiling is assigned to this mutex.

Returns 0 if successful; otherwise it returns an error code:

FOSA_EINVAL: the value of prioceiling is invalid

FOSA_EAGAIN: the system lacked the necessary resources to create the mutex

FOSA_ENOMEM: Insufficient memory exists to initialize the mutex

FOSA_EBUSY: The system has detected an attempt to reinitialize the mutex

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.5.2.2 int fosa_mutex_destroy (fosa_mutex_t ∗ *mutex*)

fosa_mutex_destroy()

Destroy a frsh mutex

The mutex pointed to by mutex is destroyed

Returns 0 if successful; otherwise it returns an error code:

FOSA_EINVAL: the value of mutex is invalid

FOSA_EBUSY: The mutex is in use (is locked)

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.5.2.3 int fosa_mutex_set_prioceiling (fosa_mutex_t * *mutex*, int *new_ceiling*, int * *old_ceiling*)

fosa_mutex_set_prioceiling()

Dynamically set the priority ceiling of a mutex

This function locks the mutex (blocking the calling thread if necessary) and after it is locked it changes its priority ceiling to the value specified by new_ceiling, and then it unlocks the mutex. The previous value of the ceiling is returned in old_ceiling.

Returns 0 if successful; otherwise it returns an error code:

FOSA_EINVAL: the value of mutex or prioceiling is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.5.2.4 int fosa_mutex_get_prioceiling (const fosa_mutex_t * *mutex*, int * *ceiling*)

fosa_mutex_get_prioceiling()

Dynamically get the priority ceiling of a mutex

This function copies into the variable pointed to by ceiling the current priority ceiling of the mutex referenced by mutex

Returns 0 if successful; otherwise it returns an error code:

FOSA_EINVAL: the value of mutex is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.5.2.5 int fosa_mutex_lock (fosa_mutex_t * *mutex*)

fosa_mutex_lock()

Lock a mutex

This function locks the mutex specified by mutex. If it is already locked, the calling thread blocks until the mutex becomes available. The operation returns with the mutex in the locked state, with the calling thread as its owner.

Returns 0 if successful; otherwise it returns an error code:

FOSA_EINVAL: the value of mutex is invalid, or the priority of the calling thread is higher than the priority ceiling of the mutex

FOSA_EDEADLK: the current thread already owns this mutex

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.5.2.6 int fosa_mutex_trylock (fosa_mutex_t * *mutex*)

fosa_mutex_trylock()

Try locking a mutex

This function is identical to fosa_mutex_lock() except that if the mutex is already locked the call returns immediately with an error indication.

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the value of mutex is invalid, or the priority of the calling thread is higher than the priority ceiling of the mutex

FOSA_EBUSY: the mutex was already locked

Alternatively, except for FOSA_EBUSY, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.5.2.7 int fosa_mutex_unlock (fosa_mutex_t * *mutex*)

fosa_mutex_unlock()

Unlock a mutex

This function must be called by the owner of the mutex referenced by mutex, to unlock it. If there are threads blocked on the mutex the mutex becomes available and the highest priority thread is awakened to acquire the mutex.

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the value of mutex is invalid FOSA_EPERM: the calling thread is not the owner of the mutex

Alternatively, except for FOSA_EBUSY, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.5.2.8 int fosa_cond_init (fosa_cond_t * *cond*)

fosa_cond_init()

Initiatize a condition variable

The condition variable referenced by cond is initialized with the attributes required by the FOSA implementation.

Returns 0 if successful; otherwise it returns an error code: FOSA_EAGAIN: the system lacked the necessary resources to create the condition variable FOSA_ENOMEM: Insufficient memory exists to initialize the condition variable FOSA_EBUSY: The system has detected an attempt to reinitialize the condition variable

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.5.2.9 int fosa_cond_destroy (fosa_cond_t * *cond*)

fosa_cond_destroy()

Destroy a condition variable

The condition variable pointed to by cond is destroyed

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the value of cond is invalid FOSA_EBUSY: The condition variable is in use (a thread is waiting on it)

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.5.2.10 int fosa_cond_signal (fosa_cond_t * *cond*)

fosa_cond_signal()

Signal a condition variable

This call unblocks at least one of the threads that are waiting on the condition variable referenced by cond. If there are no threads waiting, the function has no effect

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the value of cond is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.5.2.11 int fosa_cond_broadcast (fosa_cond_t * *cond*)

fosa_cond_broadcast()

Broadcast a condition variable

This call unblocks all of the threads that are waiting on the condition variable referenced by cond. If there are no threads waiting, the function has no effect.

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the value of cond is invalid

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.5.2.12 int fosa_cond_wait (fosa_cond_t * *cond*, fosa_mutex_t * *mutex*)

fosa_cond_wait()

Wait at a condition variable

This call is used to block on the condition variable referenced by cond. It shall be called with the mutex referenced by mutex locked. The function releases the mutex and blocks the calling thread until the condition is signalled by some other thread and the calling thread is awakened. Then it locks the mutex and returns with the mutex locked by the calling thread.

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the value of cond or mutex is invalid, or different mutexes were used for concurrent wait operations on cond, or the mutex was not owned by the calling thread

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.5.2.13 int fosa_cond_timedwait (fosa_cond_t * *cond*, fosa_mutex_t * *mutex*, const fosa_abs_time_t * *abstime*)

fosa_cond_timedwait()

Wait at a condition variable, with a timeout

This function is equal to fosa_cond_wait(), except that the maximum wait time is limited to the absolute time referenced by abstime, as measured by the FOSA_CLOCK_ABSOLUTE clock.

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the value of cond or mutex or abstime is invalid, or different mutexes were used for concurrent wait operations on cond, or the mutex was not owned by the calling thread FOSA_ETIMEDOUT: the timeout expired

Alternatively, except for FOSA_ETIMEDOUT, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

## 4.6 Thread and Signals

### Data Structures

- union fosa_signal_info_t

### Typedefs

- typedef FOSA_THREAD_ID_T_OPAQUE fosa_thread_id_t
- typedef FOSA_THREAD_ATTR_T_OPAQUE fosa_thread_attr_t
- typedef FOSA_SIGNAL_T_OPAQUE **fosa_signal_t**
- typedef void *(* fosa_thread_code_t )(void *)

### Functions

- bool fosa_thread_equal (fosa_thread_id_t t1, fosa_thread_id_t t2)
- fosa_thread_id_t fosa_thread_self ()
- int fosa_thread_attr_init (fosa_thread_attr_t *attr)
- int fosa_thread_attr_destroy (fosa_thread_attr_t *attr)
- int fosa_thread_attr_set_stacksize (fosa_thread_attr_t *attr, size_t stacksize)
- int fosa_thread_attr_get_stacksize (const fosa_thread_attr_t *attr, size_t *stacksize)
- int fosa_thread_create (fosa_thread_id_t *tid, const fosa_thread_attr_t *attr, fosa_thread_code_t code, void *arg)
- int fosa_key_create (int *key)
- int fosa_key_destroy (int key)
- int fosa_thread_set_specific_data (int key, fosa_thread_id_t tid, const void *value)
- int fosa_thread_get_specific_data (int key, fosa_thread_id_t tid, void **value)
- int fosa_get_priority_max ()
- int fosa_get_priority_min ()
- int fosa_thread_attr_set_prio (fosa_thread_attr_t *attr, int prio)
- int fosa_thread_attr_get_prio (const fosa_thread_attr_t *attr, int *prio)
- int fosa_thread_set_prio (fosa_thread_id_t tid, int prio)
- int fosa_thread_get_prio (fosa_thread_id_t tid, int *prio)
- int fosa_set_accepted_signals (fosa_signal_t set[ ], int size)
- int fosa_signal_queue (fosa_signal_t signal, fosa_signal_info_t info, fosa_thread_id_t receiver)
- int fosa_signal_wait (fosa_signal_t set[ ], int size, fosa_signal_t *signal_received, fosa_signal_info_t *info)
- int fosa_signal_timedwait (fosa_signal_t set[ ], int size, fosa_signal_t *signal_received, fosa_signal_info_t *info, const fosa_rel_time_t *timeout)

#### 4.6.1 Detailed Description

This module defines the functions that manipulate fosa_threads and fosa_signals inside FRSH implementation.

Applications can refer to FRSH threads but they cannot create them directly, instead they must use frsh_thread_create*() which in turn use fosa_thread_create().

For signals, we assume that the OS provides a direct mapping for fosa_signal_t and fosa_signal_info_t in the native interface.

#### 4.6.2 Typedef Documentation

##### 4.6.2.1 typedef FOSA_THREAD_ID_T_OPAQUE fosa_thread_id_t

identifier of a FOSA thread

#### 4.6.2.2 typedef FOSA_THREAD_ATTR_T_OPAQUE fosa_thread_attr_t

thread attributes object

#### 4.6.2.3 typedef void*(* fosa_thread_code_t)(void *)

The type references a function that may become a thread's code

### 4.6.3 Function Documentation

#### 4.6.3.1 bool fosa_thread_equal (fosa_thread_id_t *t1*, fosa_thread_id_t *t2*)

fosa_thread_equal()

Compare two thread identifiers to determine if they refer to the same thread

#### 4.6.3.2 fosa_thread_id_t fosa_thread_self ()

fosa_thread_self()

Return the thread id of the calling thread

#### 4.6.3.3 int fosa_thread_attr_init (fosa_thread_attr_t * *attr*)

fosa_thread_attr_init()

Initialize a thread attributes object

This function initializes the object pointed to by attr to all the default values defined by FRSH

**Returns:**

0 if successful; otherwise it returns
FOSA_ENOMEM: insufficient memory exists to initialize the thread attributes object

#### 4.6.3.4 int fosa_thread_attr_destroy (fosa_thread_attr_t * *attr*)

fosa_thread_attr_destroy()

Destroy a thread attributes object

This function is used to destroy the thread attributes object, pointed to by attr, and deallocate any system resources allocated for it

Returns 0

#### 4.6.3.5 int fosa_thread_attr_set_stacksize (fosa_thread_attr_t * *attr*, size_t *stacksize*)

fosa_thread_attr_set_stacksize()

Set the thread minimum stack size in a thread attributes object

This function sets the minimum stack size of the thread attributes object attr to the value given by stacksize, in bytes. This function has no runtime effect on the stack size, except when the attributes object is used to create a thread, when it will be created with the specified minimum stack size

**Returns:**

0 if successful, or the following error code: FOSA_EINVAL: the specified stacksize value is not supported in this implementation

#### 4.6.3.6 int fosa_thread_attr_get_stacksize (const fosa_thread_attr_t * *attr*, size_t * *stacksize*)

fosa_thread_attr_get_stacksize()

Get the thread minimum stack size from a thread attributes object

This function sets the variable pointed to by stacksize to the minimum stack size stored in the thread attributes object attr.

**Returns:**
0

#### 4.6.3.7 int fosa_thread_create (fosa_thread_id_t * *tid*, const fosa_thread_attr_t * *attr*, fosa_thread_code_t *code*, void * *arg*)

fosa_thread_create()

This function creates a new thread using the attributes specified in attr. If attr is NULL, default attributes are used. The new thread starts running immediately, executing the function specified by code, with an argument equal to arg. Upon successful return, the variable pointed to by tid will contain the identifier of the newly created thread. The set of signals that may be synchronously accepted is inherited from the parent thread.

Returns 0 if successful; otherwise it returs a code error:

FOSA_EAGAIN: the system lacks the necessary resources to create a new thread or the maximum number of threads has been reached

FOSA_EINVAL: the value specified by attr is invalid (for instance, it has not been correctly initialized)

FOSA_EREJECT: the cretion of the thread was rejected by the frsh scheduler possibly because of incorrect attributes, or because the requested minimum capacity cannot be guaranteed

#### 4.6.3.8 int fosa_key_create (int * *key*)

fosa_key_create()

Create a new key for thread specific data.

Prior to setting data in a key, we need ask the system to create one for us. The thread specific data of all the threads is set to the value NULL until changed to a different value via fosa_thread_set_specific_data().

**Returns:**
0 if successful
FOSA_EINVAL If we already have reached the FOSA_MAX_KEYS limit. FOSA_ENOMEM If there are no enough memory resources to create the key.

#### 4.6.3.9 int fosa_key_destroy (int *key*)

fosa_key_destroy()

Destroy a key

This destroys the key and disables its use in the system

**Returns:**
0 if successful
FOSA_EINVAL The key is not initialised or is not in FOSA key range.

#### 4.6.3.10 int fosa_thread_set_specific_data (int *key*, fosa_thread_id_t *tid*, const void * *value*)

fosa_thread_set_specific_data()

Set thread-specific data

For the thread identified by tid, the thread-specifid data field identified by key will be set to the value specified by value

Returns 0 if successful; otherwise, an error code is returned FOSA_EINVAL: the value of key is not between 0 and FOSA_MAX_KEYS-1

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.6.3.11 int fosa_thread_get_specific_data (int *key*, fosa_thread_id_t *tid*, void ** *value*)

fosa_thread_get_specific_data()

Get thread-specific data

For the thread identified by tid, the thread-specifid data field identified by key will be copied to the variable pointed to by value

Returns 0 if successful; otherwise, an error code is returned FOSA_EINVAL: the value of key is not between 0 and FOSA_MAX_KEYS-1

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.6.3.12 int fosa_get_priority_max ()

fosa_get_priority_max()

Return the maximum priority value used in this implementation

#### 4.6.3.13 int fosa_get_priority_min ()

fosa_get_priority_min()

Return the minimum priority value used in this implementation

#### 4.6.3.14 int fosa_thread_attr_set_prio (fosa_thread_attr_t * *attr*, int *prio*)

fosa_thread_attr_set_prio()

Change the priority of a thread attributes object

The priority of the thread attriutes object specified by attr is set to the value specified by prio. This function has no runtime effect on the priority, except when the attributes object is used to create a thread, when it will be created with the specified priority

Returns 0 if successful, or the following error code: FOSA_EINVAL: the specified priority value is not between the minimum and the maximum priorities defined in this FRSH implementation Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.6.3.15 int fosa_thread_attr_get_prio (const fosa_thread_attr_t * *attr*, int * *prio*)

fosa_thread_attr_get_prio()

Get the priority from a thread attributes object

This function sets the variable pointed to by prio to the priority stored in the thread attributes object attr.

Returns 0

#### 4.6.3.16 int fosa_thread_set_prio (fosa_thread_id_t *tid*, int *prio*)

fosa_thread_set_prio()

Dynamically change the priority of a thread

The priority of the thread identified by tid is set to the value specified by prio.

Returns 0 if successful, or the following error code: FOSA_EINVAL: the specified priority value is not between the minimum and the maximum priorities defined in this FRSH implementation Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.6.3.17 int fosa_thread_get_prio (fosa_thread_id_t *tid*, int * *prio*)

fosa_thread_get_prio()

Dynamically get the priority of a thread

This function sets the variable pointed to by prio to the priority of the thread identified by tid

Returns 0

#### 4.6.3.18 int fosa_set_accepted_signals (fosa_signal_t *set*[ ], int *size*)

fosa_set_accepted_signals()

Establish the set of signals that may be synchronously accepted by the calling thread

The function uses the array of signal numbers specified by set, which must be of size equal to size

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the array contains one or more values which are not between FOSA_SIGNAL_MIN and FOSA_SIGNAL_MAX, or size is less than 0

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.6.3.19 int fosa_signal_queue (fosa_signal_t *signal*, fosa_signal_info_t *info*, fosa_thread_id_t *receiver*)

fosa_signal_queue()

Queue a signal

This function is used to explicitly send a signal with a specified value

The signal number specified by signal is sent together with the information specified by info, to the thread identified by receiver. In those implementations that do not support queueing a signal with information to a thread (such as POSIX), the signal may be sent to any thread that is waiting for this signal via fosa_signal_wait(). Portability can be ensured by having the receiver thread be the one who is waiting for the signal.

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the signal specified by signal is not between FOSA_SIGNAL_MIN and FOSA_SIGNAL_MAX

FOSA_EAGAIN: no resources are available to queue the signal; the maximum number of queued signals has been reached, or a systemwide resource limit has been exceeded

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.6.3.20 int fosa_signal_wait (fosa_signal_t *set*[ ], int *size*, fosa_signal_t * *signal_received*, fosa_signal_info_t * *info*)

fosa_signal_wait()

Wait for a signal

The function waits for the arrival of one of the signals in the array of signal numbers specified by set, which must be of size equal to size. If there is a signal already queued, the function returns immediately. If there is no signal of the specified set queued, the calling thread is suspended until a signal from that set arrives. Upon return, if signal_received is not NULL the number of the signal received is stored in the variable pointed to by signal_received; and if info is not NULL the associated information is stored in the variable pointed to by info.

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the array contains one or more values which are not between FOSA_SIGNAL_MIN and FOSA_SIGNAL_MAX, or size is less than 0

Alternatively, in case of error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

#### 4.6.3.21 int fosa_signal_timedwait (fosa_signal_t *set*[ ], int *size*, fosa_signal_t * *signal_received*, fosa_signal_info_t * *info*, const fosa_rel_time_t * *timeout*)

fosa_signal_timedwait()

Timed wait for a signal

This function behaves the same as fosa_signal_wait(), except that the suspension time is limited to the time interval specified in the fosa_rel_time_t value referenced by timeout.

Returns 0 if successful; otherwise it returns an error code: FOSA_EINVAL: the array contains one or more values which are not between FOSA_SIGNAL_MIN and FOSA_SIGNAL_MAX, or size is less than 0, or timeout is invalid FOSA_EAGAIN: The timeout expired

Alternatively, in case of the FOSA_EINVAL error the implementation is allowed to notify it to the system console and then terminate the FRSH implementation and dependant applications

# 5 FOSA Data Structure Documentation

## 5.1 fosa_ads_scheduler_ops_t Struct Reference

`#include <fosa_types.h>`

### Data Fields

- void(* **init** )(void *sched_data, void *arg)
- void(* **new_thread** )(void *sched_data, fosa_thread_id_t thread, fosa_ads_actions_t *actions, fosa_abs_time_t *current_time)
- void(* **thread_terminate** )(void *sched_data, fosa_thread_id_t thread, fosa_ads_actions_t *actions, fosa_abs_time_t *current_time)

- void(∗ **thread_ready** )(void ∗sched_data, fosa_thread_id_t thread, fosa_ads_actions_t ∗actions, fosa_abs_time_t ∗current_time)
- void(∗ **thread_block** )(void ∗sched_data, fosa_thread_id_t thread, fosa_ads_actions_t ∗actions, fosa_abs_time_t ∗current_time)
- void(∗ **change_sched_param_thread** )(void ∗sched_data, fosa_thread_id_t thread, fosa_ads_actions_t ∗actions, fosa_abs_time_t ∗current_time)
- void(∗ **explicit_call_with_data** )(void ∗sched_data, fosa_thread_id_t thread, const void ∗msg, size_t msg_size, void ∗reply, size_t ∗reply_size, fosa_ads_actions_t ∗actions, fosa_abs_time_t ∗current_time)
- void(∗ **notification_for_thread** )(void ∗sched_data, fosa_thread_id_t thread, fosa_clock_id_t clock, fosa_ads_actions_t ∗actions, fosa_abs_time_t ∗current_time)
- void(∗ **timeout** )(void ∗sched_data, fosa_ads_actions_t ∗actions, fosa_abs_time_t ∗current_time)
- void(∗ **signal** )(void ∗sched_data, fosa_signal_t signal, fosa_signal_info_t siginfo, fosa_ads_actions_t ∗actions, fosa_abs_time_t ∗current_time)
- void(∗ **appsched_error** )(void ∗sched_data, fosa_thread_id_t thread, fosa_ads_error_cause_t cause, fosa_ads_actions_t ∗actions)

### 5.1.1 Detailed Description

Scheduler primitive operations

This structure is used to create application schedulers. It contains pointers to the primitive operations that are invoked by the system when a scheduling event occurs:

- The **init()** primitive operation is invoked by the system just after the scheduler has been created using fosa_ads_scheduler_create().

- The **new_thread()** primitive operation is invoked by the system when a thread has requested attachment to this scheduler; this can be a newly created thread (via fosa_thread_create()), or an existing thread that was not running under ads scheduler (via fosa_ads_set_appscheduled()).

  The thread can be rejected by the scheduler adding a reject-thread action to the actions parameter using fosa_ads_actions_add_reject(). If no reject-thread action is added, the thread is accepted.

  Newly created threads shall be activated by the system after the execution of the new_thread() primitive operation. The urgency of an accepted thread (either newly created or existing) shall be set to a value of zero, unless an activate-thread action with a different value of urgency is added via fosa_ads_actions_add_activate().

  If a request to attach a thread to this scheduler was made via fosa_ads_set_appscheduled(), at the finalization of the new_thread() primitive operation if the newly attached thread is blocked by the system (not by the scheduler itself via a suspend scheduling action), a thread-block event shall be generated for the scheduler immediately and, consequently, the thread_block() primitive operation shall be invoked by the system.

- The **thread_terminate()** primitive operation is invoked by the system when a thread attached to this scheduler is terminating (via an explicit or implicit thread termination, or cancellation, or when it is no longer sceduled by the ads scheduler (via fosa_ads_setappscheduled()).

  Before the thread_terminate() primitive operation is invoked by the system, all the thread-notification events programmed for that thread are cancelled.

  In the case of a thread that is terminating, the thread_terminate() primitive operation is executed before the execution of the cleanup handlers and of the thread-specific data destructor functions. In that way, the thread parameter corresponds to a valid thread Id and the thread-specific data is valid and can be accessed from the thread_terminate() primitive operation.

Also for terminating threads, after the thread_terminate() primitive operation finishes, the system shall lower the urgency of the thread identified by thread to a value of zero, and shall deattach it from the ads scheduler. Then, the thread shall execute the cleanup handlers and the thread-specific data destructor functions outside the management of its former scheduler. Notice that in a multiprocessor system this may imply the suspension of the thread identified by parameter thread during the execution of the thread_terminate() primitive operation.

- The **thread_ready()** primitive operation is invoked by the system when a thread attached to this scheduler that was blocked has become unblocked by the system.

- The **thread_block()** primitive operation is invoked by the system when a thread attached to this scheduler has blocked.

- The **change_sched_param_thread()** primitive operation is invoked by the system when the scheduling parameters of a thread attached to this scheduler have been changed OUTSIDE OF AN SCHEDULER CALLBACK, and the thread continues to run under this scheduler. The change includes either the regular scheduling parameters (fosa_thread_set_prio() or the application-defined scheduling parameters, via fosa_ads_set_appsched_param().

- The **explicit_call_with_data()** primitive operation is invoked by the system when a thread (identified by the thread parameter) has explicitly invoked the scheduler with a message containing scheduling information, and possibly requesting a reply message, via fosa_ads_invoke_withdata().

- The **notification_for_thread()** primitive operation is invoked by the system when the time for a thread-notification previously programed by the scheduler via fosa_ads_actions_add_thread_notification() is reached. Parameter clock identifies the clock for which the thread-notification was programmed.

- The **timeout()** primitive operation is invoked by the system when a timeout requested by the scheduler (via fosa_ads_actions_add_timeout()) has expired.

- The **signal()** primitive operation is invoked by the system when a signal belonging to the set of signals for which the scheduler is waiting (via fosa_ads_set_handled_signal_set()) has been generated.

  The signal number and its associated information (if any) are passed in the arguments signal and siginfo.

  The signal is consumed with the invocation of this primitive operation, which implies that it will not cause the execution of any signal handler, nor it may be accepted by any thread waiting for this signal number.

- The **appsched_error()** primitive operation is invoked by the system when an error in the scheduling actions list specified in a previous primitive operation is detected. The cause of the error is notified in the parameter cause. The defined causes of error are described fosa_ads_error_cause_t

Every primitive operation receives the argument sched_data. It is a pointer to a memory area containing information shared by all the scheduler operations. It can be used to store the data structures required by the scheduler (for example, a ready queue and a delay queue). Scheduler operations should not use any other global data out of this memory area.

The actions argument is used by the scheduler to request the operating system to execute a set of scheduling actions at the end of the primitive operation. It is passed empty by the system, and the scheduler may add multiple scheduling actions.

The current_time argument contains the system time measured immediately before the invocation of the primitive operation using the FOSA_CLOCK_REALTIME clock

In addition to these common parameters, most of the primitive operations receive a thread argument. This argument allows the primitive operations to know which is the thread that has produced or is related to the event.

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

- include/fosa_types.h

## 5.2 fosa_signal_info_t Union Reference

`#include <fosa_types.h>`

### Data Fields

- int **sival_int**
- void * **sival_ptr**

### 5.2.1 Detailed Description

information associated to a signal

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

- include/fosa_types.h

# Index