

My Project

Generated by Doxygen 1.8.6

Thu Jul 21 2016 09:38:43

Contents

1	README	1
2	Module Index	3
2.1	Modules	3
3	Module Documentation	5
3.1	Library functions	5
3.1.1	Detailed Description	5
3.1.2	Function Documentation	5
3.1.2.1	glt_end	5
3.1.2.2	glt_finalize	5
3.1.2.3	glt_init	5
3.1.2.4	glt_start	6
3.2	Barrier functions	7
3.2.1	Detailed Description	7
3.2.2	Function Documentation	7
3.2.2.1	glt_barrier_create	7
3.2.2.2	glt_barrier_free	7
3.2.2.3	glt_barrier_wait	7
3.3	Condition functions	8
3.3.1	Detailed Description	8
3.3.2	Function Documentation	8
3.3.2.1	glt_cond_broadcast	8
3.3.2.2	glt_cond_create	8
3.3.2.3	glt_cond_free	8
3.3.2.4	glt_cond_signal	8
3.3.2.5	glt_cond_wait	9
3.4	Mutex functions	10
3.4.1	Detailed Description	10
3.4.2	Function Documentation	10
3.4.2.1	glt_mutex_create	10
3.4.2.2	glt_mutex_free	10

3.4.2.3	glt_mutex_lock	10
3.4.2.4	glt_mutex_trylock	10
3.4.2.5	glt_mutex_unlock	11
3.5	Work-units functions	12
3.5.1	Detailed Description	12
3.5.2	Function Documentation	12
3.5.2.1	glt_tasklet_cancel	12
3.5.2.2	glt_tasklet_create	13
3.5.2.3	glt_tasklet_create_to	13
3.5.2.4	glt_tasklet_join	13
3.5.2.5	glt_tasklet_malloc	13
3.5.2.6	glt_tasklet_self	13
3.5.2.7	glt_ult_cancel	15
3.5.2.8	glt_ult_create	15
3.5.2.9	glt_ult_create_to	15
3.5.2.10	glt_ult_exit	15
3.5.2.11	glt_ult_get_id	15
3.5.2.12	glt_ult_join	16
3.5.2.13	glt_ult_malloc	16
3.5.2.14	glt_ult_migrate_self_to	16
3.5.2.15	glt_ult_self	16
3.5.2.16	glt_workunit_get_thread_id	16
3.5.2.17	glt_yield	17
3.5.2.18	glt_yield_to	17
3.6	Timer functions	18
3.6.1	Detailed Description	18
3.6.2	Function Documentation	18
3.6.2.1	glt_get_wtime	18
3.6.2.2	glt_timer_create	18
3.6.2.3	glt_timer_free	18
3.6.2.4	glt_timer_get_secs	19
3.6.2.5	glt_timer_start	19
3.6.2.6	glt_timer_stop	19
3.7	Util functions	20
3.7.1	Detailed Description	20
3.7.2	Function Documentation	20
3.7.2.1	glt_get_num_threads	20
3.7.2.2	glt_get_thread_num	20
3.8	Scheduler functions	21
3.8.1	Detailed Description	21

3.8.2	Function Documentation	21
3.8.2.1	glt_scheduler_create_basic	21
3.8.2.2	glt_scheduler_config_free	22
3.8.2.3	glt_scheduler_create	22
3.8.2.4	glt_scheduler_exit	22
3.8.2.5	glt_scheduler_finish	22
3.8.2.6	glt_scheduler_free	22
3.8.2.7	glt_scheduler_get_data	23
3.8.2.8	glt_scheduler_get_size	23
3.8.2.9	glt_scheduler_get_total_size	23
3.8.2.10	glt_scheduler_has_to_stop	23
3.8.2.11	glt_scheduler_set_data	23
3.9	Key functions	25
3.9.1	Detailed Description	25
3.9.2	Function Documentation	25
3.9.2.1	glt_key_create	25
3.9.2.2	glt_key_free	25
3.9.2.3	glt_key_get	25
3.9.2.4	glt_key_set	25
3.10	GLT object list	27
3.10.1	Detailed Description	27
Index		28

Chapter 1

README

GLT (Generic Lightweight Threads). Common API for Lightweight Thread Implementations.

- Developed by:
 - Adrian Castello (adcastel@uji.es) at Universitat Jaume I
- Supervised by:
 - Antonio J. Peña (antonio.pena@bsc.es) at Barcelona Supercomputing Center
 - Rafael Mayo Gual and Enrique S. Quintana-Ortí ({mayo,quintana}@uji.es)
 - Sangmin Seo and Pavan Balaji ({sseo,balaji}@anl.gov) at Argonne National Laboratory

GLT Release 2.5

GLT is a common API for HPC lightweight thread (LWT) libraries. It supports MassiveThreads, Qthreads, and Argobots as underlying LWT solutions. Moreover, GLT over Pthread is implemented with comparative purpose.

In addition, GLT can be used as POSIX threads API since version 2.5.

1. Getting Started
2. How to use GLT
3. How to cite GLT
4. Reporting Problems

1. Getting Started

The following instructions take you through a sequence of steps to get GLT installed and compiled.

(a.1) You will need the following prerequisites.

- REQUIRED: This tar file `GLT-2.5.tar.gz`
- REQUIRED: A C compiler (`gcc` is sufficient)

(a.2) At least one of these libraries:

- Argobots library.
- Qthreads library.
- MassiveThreads library.

(b) Unpack the tar file and go to the top level directory:

```
tar xzf GLT-2.5.tar.gz
cd GLT
```

If your tar doesn't accept the z option, use

```
gunzip GLT-2.5.tar.gz
tar xf GLT-2.5.tar
cd GLT
```

(c) Define environment variables:

The definition of the HOME_ARG, HOME_QTH, and HOME_MTH environment variables with the path to Argobots, Qthreads, and MassiveThreads libraries respectively is required.

(d) Build GLT:

```
cd src

for csh and tcsh:

    make [arg|qth|mth|pth] |& tee m.txt

for bash and sh:

    make [arg|qth|mth|pth] 2>&1 | tee m.txt
```

2. How to use GLT

I. GLT offers two library approaches:

(a) Dynamic library. Once the step 1 is completed, a libglt.so file can be found in each underlying library folder. The glt.h file needs to be included in the user's code and the -lglt flag added to the compilation order.

(d) Static library. In order to use this performance-oriented implementation fast_glt.h file may be included in the user's code and the -DFASTGLT flag added to the compilation order.

II. Using Pthreads API with GLT

GLT also offers the use of code written with pthreads just including "glt_pthreads.h" instead of "pthread.h"

3. How to cite GLT

To cite GLT in your work, please use the following for now: A. Castelló, A.J. Peña, S. Seo, R. Mayo, P. Balaji, E.S. Quintana-Ortí. GLT: A common API for lightweight thread libraries. www.hpca.uji.es/GLT. 2016

4. Reporting Problems

If you have problems with the installation or usage of GLT, please send an email to adcastel@uji.es.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Library functions	5
Barrier functions	7
Condition functions	8
Mutex functions	10
Work-units functions	12
Timer functions	18
Util functions	20
Scheduler functions	21
Key functions	25
GLT object list	27

Chapter 3

Module Documentation

3.1 Library functions

Functions

- void `glt_start` (void)
Entry point for the GLT dynamic library.
- void `glt_end` (void)
Ending point for the GLT dynamic library.
- void `glt_init` (int argc, char *argv[])
GLT initialization function.
- void `glt_finalize` ()
GLT finalization function.

3.1.1 Detailed Description

These functions start/stop and open/close the underlying GLT libraries. They are used in dynamic and static implementations.

3.1.2 Function Documentation

3.1.2.1 void `glt_end` (void)

Ending point for the GLT dynamic library.

`glt_end()` is the last called function when the GLT dynamic library is unloaded

3.1.2.2 void `glt_finalize` ()

GLT finalization function.

`glt_finalize()` destroys the GLT environment. It is not mandatory and should be the last GLT function call.

3.1.2.3 void `glt_init` (int *argc*, char * *argv*[])

GLT initialization function.

`glt_init()` sets the GLT environment up. It is mandatory and needs to be the first GLT function call.

Parameters

in	<i>argc</i>	
in	<i>argv</i>	

3.1.2.4 void glt_start (void)

Entry point for the GLT dynamic library.

`glt_start\(\)` is the first called function when the GLT dynamic library is loaded

3.2 Barrier functions

Functions

- void `glt_barrier_create` (int num_waiters, GLT_barrier *barrier)
Creates a barrier.
- void `glt_barrier_free` (GLT_barrier *barrier)
Destroys a barrier.
- void `glt_barrier_wait` (GLT_barrier *barrier)
Waits into a barrier.

3.2.1 Detailed Description

These functions manage the GLT barriers for the ULTs.

3.2.2 Function Documentation

3.2.2.1 void glt_barrier_create (int num_waiters, GLT_barrier * barrier)

Creates a barrier.

`glt_barrier_create()` creates a barrier for ULTs.

Parameters

in	<i>num_waiters</i>	Indicates the number of ULTs requested to continue
in, out	<i>barrier</i>	Handle to newly created GLT_barrier

3.2.2.2 void glt_barrier_free (GLT_barrier * barrier)

Destroys a barrier.

`glt_barrier_free()` destroys a barrier for ULTs.

Parameters

in	<i>barrier</i>	Handle to the target GLT_barrier.
----	----------------	-----------------------------------

3.2.2.3 void glt_barrier_wait (GLT_barrier * barrier)

Waits into a barrier.

`glt_barrier_wait()` Executed by a ULT, it waits until the number of waiters is achieved.

Parameters

in	<i>barrier</i>	Handle to the target GLT_barrier.
----	----------------	-----------------------------------

3.3 Condition functions

Functions

- void `glt_cond_create` (`GLT_cond *cond`)
Creates a condition.
- void `glt_cond_free` (`GLT_cond *cond`)
Destroys a condition.
- void `glt_cond_signal` (`GLT_cond cond`)
Sends a signal for a condition.
- void `glt_cond_wait` (`GLT_cond cond`, `GLT_mutex mutex`)
A ULT waits in this point for a signal.
- void `glt_cond_broadcast` (`GLT_cond cond`)
Broadcast a signal for a condition.

3.3.1 Detailed Description

These functions manage the GLT conditions for the ULTs.

3.3.2 Function Documentation

3.3.2.1 void `glt_cond_broadcast` (`GLT_cond cond`)

Broadcast a signal for a condition.

`glt_cond_broadcast()` broadcasts a signal for ULTs.

Parameters

in	<code>cond</code>	Handle to the target GLT_condition.
----	-------------------	-------------------------------------

3.3.2.2 void `glt_cond_create` (`GLT_cond * cond`)

Creates a condition.

`glt_cond_create()` creates a condition for ULTs.

Parameters

in, out	<code>cond</code>	Handle to newly created GLT_condition
---------	-------------------	---------------------------------------

3.3.2.3 void `glt_cond_free` (`GLT_cond * cond`)

Destroys a condition.

`glt_cond_free()` destroys a condition for ULTs.

Parameters

in	<code>cond</code>	Handle to the target GLT_condition.
----	-------------------	-------------------------------------

3.3.2.4 void `glt_cond_signal` (`GLT_cond cond`)

Sends a signal for a condition.

`glt_cond_signal()` sends a signal for ULTs.

Parameters

in	<i>cond</i>	Handle to the target GLT_condition.
----	-------------	-------------------------------------

3.3.2.5 void glt_cond_wait (GLT_cond *cond*, GLT_mutex *mutex*)

A ULT waits in this point for a signal.

`glt_cond_wait()` a ULT waits at this point for a signal to access the mutex.

Parameters

in	<i>cond</i>	Handle to the target GLT_condition.
in	<i>mutex</i>	Handle to the target GLT_mutex.

3.4 Mutex functions

Functions

- void `glt_mutex_create` (`GLT_mutex *mutex`)
Creates a mutex.
- void `glt_mutex_lock` (`GLT_mutex mutex`)
Locks a mutex.
- void `glt_mutex_unlock` (`GLT_mutex mutex`)
Unlocks a mutex.
- void `glt_mutex_free` (`GLT_mutex *mutex`)
Destroys a mutex.
- void `glt_mutex_trylock` (`GLT_bool *locked`, `GLT_mutex mutex`)
Tries to lock a mutex.

3.4.1 Detailed Description

These functions manage the GLT mutexes for the ULTs.

3.4.2 Function Documentation

3.4.2.1 void `glt_mutex_create` (`GLT_mutex * mutex`)

Creates a mutex.

`glt_mutex_create()` creates a mutex for ULTs.

Parameters

<code>in, out</code>	<code>mutex</code>	Handle to newly created <code>GLT_mutex</code>
----------------------	--------------------	------------------------------------------------

3.4.2.2 void `glt_mutex_free` (`GLT_mutex * mutex`)

Destroys a mutex.

`glt_mutex_free()` destroys a mutex for ULTs.

Parameters

<code>in</code>	<code>mutex</code>	Handle to the target <code>GLT_mutex</code> .
-----------------	--------------------	-----------------------------------------------

3.4.2.3 void `glt_mutex_lock` (`GLT_mutex mutex`)

Locks a mutex.

`glt_mutex_lock()` locks (if possible) a mutex.

Parameters

<code>in</code>	<code>mutex</code>	Handle to the target <code>GLT_mutex</code> .
-----------------	--------------------	-----------------------------------------------

3.4.2.4 void `glt_mutex_trylock` (`GLT_bool * locked`, `GLT_mutex mutex`)

Tries to lock a mutex.

`glt_mutex_trylock()` tries to lock a mutex.

Parameters

in	<i>mutex</i>	Handle to the target GLT_mutex.
out	<i>locked</i>	GLT_bool with the value 1 if the mutex has been locked or 0 if it was not possible.

3.4.2.5 void glt_mutex_unlock (GLT_mutex mutex)

Unlocks a mutex.

`glt_mutex_unlock()` unlocks a mutex.

Parameters

in	<i>mutex</i>	Handle to the target GLT_mutex.
----	--------------	---------------------------------

3.5 Work-units functions

Functions

- `GLT_ult * glt_ult_malloc (int number_of_ult)`
ULT allocation.
- `GLT_tasklet * glt_tasklet_malloc (int number_of_tasklets)`
ULT allocation.
- `void glt_ult_create (void(*thread_func)(void *), void *arg, GLT_ult *new_ult)`
ULT creation.
- `void glt_ult_create_to (void(*thread_func)(void *), void *arg, GLT_ult *new_ult, int dest)`
ULT creation in a given destination.
- `void glt_tasklet_create (void(*thread_func)(void *), void *arg, GLT_tasklet *new_ult)`
Tasklet creation.
- `void glt_tasklet_create_to (void(*thread_func)(void *), void *arg, GLT_tasklet *new_ult, int dest)`
Tasklet creation.
- `void glt_yield ()`
ULT yields to another ready ULT.
- `void glt_yield_to (GLT_ult ult)`
ULT yields to an specific ULT.
- `void glt_ult_join (GLT_ult *ult)`
Joins an specific ULT.
- `void glt_tasklet_join (GLT_tasklet *tasklet)`
Joins an specific Tasklet.
- `void glt_ult_get_id (GLT_ult_id *id, GLT_ult ult)`
Return the unique id of a ULT.
- `void glt_workunit_get_thread_id (GLT_thread_id *id)`
Return the unique id of a thread.
- `void glt_ult_migrate_self_to (GLT_thread_id id)`
Migrates the current ULT to another thread ready queue.
- `void glt_ult_self (GLT_ult *ult)`
Obtains the current ULT handle.
- `void glt_tasklet_self (GLT_tasklet *tasklet)`
Obtains the current Tasklet handle.
- `void glt_ult_cancel (GLT_ult ult)`
Cancels an specific ULT.
- `void glt_tasklet_cancel (GLT_tasklet tasklet)`
Cancels an specific Tasklet.
- `void glt_ult_exit ()`
Exits the current ULT.

3.5.1 Detailed Description

These functions create, map, schedule, join, and execute work-units.

3.5.2 Function Documentation

3.5.2.1 void glt_tasklet_cancel (GLT_tasklet tasklet)

Cancels an specific Tasklet.

`glt_tasklet_cancel()` cancels a given `GLT_tasklet`.

Parameters

in	<i>tasklet</i>	Handle to the target GLT_tasklet.
----	----------------	-----------------------------------

3.5.2.2 void glt_tasklet_create (void(*)(void *) *thread_func*, void * *arg*, GLT_tasklet * *new_ult*)

Tasklet creation.

`glt_tasklet_create()` creates a GLT_tasklet.

Parameters

in	<i>thread_func</i>	Is the function pointer to be executed by the GLT_tasklet.
in	<i>arg</i>	Are the arguments for <i>thread_func</i> .
out	<i>new_ult</i>	Handle to a newly created GLT_tasklet.

3.5.2.3 void glt_tasklet_create_to (void(*)(void *) *thread_func*, void * *arg*, GLT_tasklet * *new_ult*, int *dest*)

Tasklet creation.

`glt_tasklet_create()` creates a GLT_tasklet.

Parameters

in	<i>thread_func</i>	Is the function pointer to be executed by the GLT_tasklet.
in	<i>arg</i>	Are the arguments for <i>thread_func</i> .
out	<i>new_ult</i>	Handle to a newly created GLT_tasklet.
in	<i>dest</i>	Number of the GLT_thread that should execute the newly created GLT_tasklet.

3.5.2.4 void glt_tasklet_join (GLT_tasklet * *tasklet*)

Joins an specific Tasklet.

`glt_tasklet_join()` joins a given GLT_tasklet.

Parameters

in	<i>tasklet</i>	Handle to the target GLT_tasklet.
----	----------------	-----------------------------------

3.5.2.5 GLT_tasklet* glt_tasklet_malloc (int *number_of_tasklets*)

ULT allocation.

`glt_tasklet_malloc()` allocates memory for a given number of GLT_tasklet.

Parameters

in	<i>number_of_tasklets</i>	Indicates the total number of GLT_tasklets that needs to be allocated.
----	---------------------------	------------------------------------------------------------------------

Returns

The pointer to the allocated memory.

3.5.2.6 void glt_tasklet_self (GLT_tasklet * *tasklet*)

Obtains the current Tasklet handle.

`glt_tasklet_self()` returns the current `GLT_tasklet` handler.

Parameters

out	<i>tasklet</i>	Handler of the the current GLT_tasklet.
-----	----------------	-----------------------------------------

3.5.2.7 void glt_ult_cancel (GLT_ult ult)

Cancels an specific ULT.

`glt_ult_cancel()` cancels a given GLT_ult.

Parameters

in	<i>ult</i>	Handle to the target GLT_ult.
----	------------	-------------------------------

3.5.2.8 void glt_ult_create (void(*)(void *) thread_func, void * arg, GLT_ult * new_ult)

ULT creation.

`glt_ult_create()` creates a GLT_ult.

Parameters

in	<i>thread_func</i>	Is the function pointer to be executed by the GLT_ult.
in	<i>arg</i>	Are the arguments for thread_func.
out	<i>new_ult</i>	Handle to a newly created GLT_ult.

3.5.2.9 void glt_ult_create_to (void(*)(void *) thread_func, void * arg, GLT_ult * new_ult, int dest)

ULT creation in a given destination.

`glt_ult_create_to()` creates a GLT_ult in a particular destination.

Parameters

in	<i>thread_func</i>	Is the function pointer to be executed by the GLT_ult.
in	<i>arg</i>	Are the arguments for thread_func.
out	<i>new_ult</i>	Handle to a newly created GLT_ult.
in	<i>dest</i>	Number of the GLT_thread that should execute the newly created GLT_ult.

3.5.2.10 void glt_ult_exit ()

Exits the current ULT.

`glt_ult_exit()` cancels the current GLT_ult.

3.5.2.11 void glt_ult_get_id (GLT_ult_id * id, GLT_ult ult)

Return the unique id of a ULT.

`glt_ult_get_id()` returns the id of a given GLT_ult.

Parameters

in	<i>ult</i>	Handle to the target GLT_ult.
out	<i>id</i>	Identifier if the the target GLT_ult.

3.5.2.12 void glt_ult_join (GLT_ult * *ult*)

Joins an specific ULT.

`glt_ult_join()` joins a given GLT_ult.

Parameters

in	<i>ult</i>	Handle to the target GLT_ult.
----	------------	-------------------------------

3.5.2.13 GLT_ult* glt_ult_malloc (int *number_of_ult*)

ULT allocation.

`glt_ult_malloc()` allocates memory for a given number of GLT_ult.

Parameters

in	<i>number_of_ult</i>	Indicates the total number of GLT_ult that needs to be allocated.
----	----------------------	-------------------------------------------------------------------

Returns

The pointer to the allocated memory.

3.5.2.14 void glt_ult_migrate_self_to (GLT_thread_id *id*)

Migrates the current ULT to another thread ready queue.

`glt_ult_migrate_self_to()` moves the current GLT_ult to another GLT_thread ready queue.

Parameters

in	<i>The</i>	identifier of the the GLT_thread destination.
----	------------	-----------------------------------------------

3.5.2.15 void glt_ult_self (GLT_ult * *ult*)

Obtains the current ULT handle.

`glt_ult_self()` returns the current GLT_ult handler.

Parameters

out	<i>ult</i>	Handler of the the current GLT_ult.
-----	------------	-------------------------------------

3.5.2.16 void glt_workunit_get_thread_id (GLT_thread_id * *id*)

Return the unique id of a thread.

`glt_workunit_get_thread_id()` returns the id of the current GLT_thread.

Parameters

out	<i>id</i>	Identifier of the the current GLT_thread.
-----	-----------	-------------------------------------------

3.5.2.17 void glt_yield ()

ULT yields to another ready ULT.

`glt_yield()` puts the current GLT_ult in the ready queue and allows another ready GLT_ult to be executed.

3.5.2.18 void glt_yield_to (GLT_ult ult)

ULT yields to an specific ULT.

`glt_yield_to()` puts the current GLT_ult in the ready queue and allows an specific ready GLT_ult to be executed.

Parameters

in	<i>ult</i>	Handle to the target GLT_ult.
----	------------	-------------------------------

3.6 Timer functions

Functions

- double `glt_get_wtime()`
Returns the current time.
- void `glt_timer_create (GLT_timer *timer)`
Creates a timer.
- void `glt_timer_free (GLT_timer *timer)`
Destroys a timer.
- void `glt_timer_start (GLT_timer timer)`
Initiates a timer.
- void `glt_timer_stop (GLT_timer timer)`
Stops a timer.
- void `glt_timer_get_secs (GLT_timer timer, double *secs)`
Obtains the elapsed time.

3.6.1 Detailed Description

These functions simplify the use of timers.

3.6.2 Function Documentation

3.6.2.1 double `glt_get_wtime()`

Returns the current time.

`glt_get_wtime()` returns the time.

Returns

The time in seconds.

3.6.2.2 void `glt_timer_create (GLT_timer * timer)`

Creates a timer.

`glt_timer_create()` creates a timer.

Parameters

<code>in, out</code>	<code>timer</code>	Handle to newly created GLT_timer.
----------------------	--------------------	------------------------------------

3.6.2.3 void `glt_timer_free (GLT_timer * timer)`

Destroys a timer.

`glt_timer_free()` destroys a timer.

Parameters

in	<i>timer</i>	Handle to the target GLT_timer.
----	--------------	---------------------------------

3.6.2.4 void glt_timer_get_secs (GLT_timer *timer*, double * *secs*)

Obtains the elapsed time.

`glt_timer_get_secs()` given a timer. It calculates the elapsed time in seconds.

Parameters

in	<i>timer</i>	Handle to the target GLT_timer.
out	<i>secs</i>	Seconds.

3.6.2.5 void glt_timer_start (GLT_timer *timer*)

Initiates a timer.

`glt_timer_start()` initiates a timer.

Parameters

in	<i>timer</i>	Handle to the target GLT_timer.
----	--------------	---------------------------------

3.6.2.6 void glt_timer_stop (GLT_timer *timer*)

Stops a timer.

`glt_timer_stop()` stops a timer.

Parameters

in	<i>timer</i>	Handle to the target GLT_timer.
----	--------------	---------------------------------

3.7 Util functions

Functions

- `int glt_get_thread_num ()`
Obtains the number of the current thread.
- `int glt_get_num_threads ()`
Returns the total number of threads.

3.7.1 Detailed Description

These functions return values from the environment set up.

3.7.2 Function Documentation

3.7.2.1 `int glt_get_num_threads ()`

Returns the total number of threads.

`glt_get_num_threads ()` returns the number threads.

Returns

The number of `GLT_threads`.

3.7.2.2 `int glt_get_thread_num ()`

Obtains the number of the current thread.

`glt_get_thread_num ()` returns the number of the current thread.

Returns

The number of the current `GLT_thread`.

3.8 Scheduler functions

Functions

- void `glt_scheduler_config_free` (`GLT_sched_config` *config)
Destroys the scheduler configuration.
- void `glt_scheduler_create` (`GLT_sched_def` *def, int num_threads, int *threads_id, `GLT_sched_config` config, `GLT_sched` *newsched)
Creates a new scheduler.
- void `glt_scheduleduler_create_basic` (`GLT_sched_predef` predef, int num_threads, int *threads_id, `GLT_sched_config` config, `GLT_sched` *newsched)
Creates a new scheduler with predefined behaviour.
- void `glt_scheduler_free` (`GLT_sched` *sched)
Destroys a scheduler.
- void `glt_scheduler_finish` (`GLT_sched` sched)
Finalizes a scheduler.
- void `glt_scheduler_exit` (`GLT_sched` sched)
Stops a scheduler.
- void `glt_scheduler_has_to_stop` (`GLT_sched` sched, `GLT_bool` *stop)
Requires to a scheduler to stop.
- void `glt_scheduler_set_data` (`GLT_sched` sched, void *data)
Sets new data to a scheduler.
- void `glt_scheduler_get_data` (`GLT_sched` sched, void **data)
gets data from a scheduler.
- void `glt_scheduler_get_size` (`GLT_sched` sched, size_t *size)
gets the current size from the scheduler.
- void `glt_scheduler_get_total_size` (`GLT_sched` sched, size_t *size)
gets the total size from the scheduler.

3.8.1 Detailed Description

These functions manages the configurable scheduler (just with Argobots).

3.8.2 Function Documentation

3.8.2.1 void `glt_scheduleduler_create_basic` (`GLT_sched_predef` predef, int num_threads, int * threads_id, `GLT_sched_config` config, `GLT_sched` * newsched)

Creates a new scheduler with predefined behaviour.

`glt_scheduleduler_create_basic()` creates a new scheduler for some threads with a predefined behaviour.

Parameters

in	<i>def</i>	Handle of the target <code>GLT_sched_predef</code> .
in	<i>num_threads</i>	number of <code>GLT_thread</code> affected by this scheduler.
in	<i>threads_id</i>	pointer to an array of <code>GLT_threads_id</code> .
in	<i>config</i>	Handle of the target <code>GLT_sched_config</code> .

out	<i>newsched</i>	Handle of new GLT_sched.
-----	-----------------	--------------------------

3.8.2.2 void glt_scheduler_config_free (GLT_sched_config * config)

Destroys the scheduler configuration.

[glt_scheduler_config_free\(\)](#) deletes the scheduler configuration.

Parameters

in	<i>config</i>	Handle of the target GLT_sched_config.
----	---------------	----------------------------------------

3.8.2.3 void glt_scheduler_create (GLT_sched_def * def, int num_threads, int * threads_id, GLT_sched_config config, GLT_sched * newsched)

Creates a new scheduler.

[glt_scheduler_create\(\)](#) creates a new scheduler for some threads.

Parameters

in	<i>def</i>	Handle of the target GLT_sched_def.
in	<i>num_threads</i>	number of GLT_thread affected by this scheduler.
in	<i>threads_id</i>	pointer to an array of GLT_threads_id.
in	<i>config</i>	Handle of the target GLT_sched_config.
out	<i>newsched</i>	Handle of new GLT_sched.

3.8.2.4 void glt_scheduler_exit (GLT_sched sched)

Stops a scheduler.

[glt_scheduler_exit\(\)](#) Stops a scheduler.

Parameters

in	<i>sched</i>	Handle of the target GLT_sched.
----	--------------	---------------------------------

3.8.2.5 void glt_scheduler_finish (GLT_sched sched)

Finalizes a scheduler.

[glt_scheduler_finish\(\)](#) finalizes a scheduler.

Parameters

in	<i>sched</i>	Handle of the target GLT_sched.
----	--------------	---------------------------------

3.8.2.6 void glt_scheduler_free (GLT_sched * sched)

Destroys a scheduler.

[glt_scheduler_free\(\)](#) destroys a scheduler.

Parameters

in	<i>sched</i>	Handle of the target GLT_sched.
----	--------------	---------------------------------

3.8.2.7 void glt_scheduler_get_data (GLT_sched sched, void ** data)

gets data from a scheduler.

`glt_scheduler_get_data()` gets data from a scheduler.

Parameters

in	<i>sched</i>	Handle of the target GLT_sched.
out	<i>data</i>	obtained.

3.8.2.8 void glt_scheduler_get_size (GLT_sched sched, size_t * size)

gets the current size from the scheduler.

`glt_scheduler_get_size()` gets size from a scheduler.

Parameters

in	<i>sched</i>	Handle of the target GLT_sched.
out	<i>size</i>	obtained.

3.8.2.9 void glt_scheduler_get_total_size (GLT_sched sched, size_t * size)

gets the total size from the scheduler.

`glt_scheduler_get_total_size()` gets the total size from a scheduler.

Parameters

in	<i>sched</i>	Handle of the target GLT_sched.
out	<i>size</i>	obtained.

3.8.2.10 void glt_scheduler_has_to_stop (GLT_sched sched, GLT_bool * stop)

Requires to a scheduler to stop.

`glt_scheduler_has_to_stop()` Requires a scheduler to stop.

Parameters

in	<i>sched</i>	Handle of the target GLT_sched.
out	<i>stop</i>	shows the answer of the scheduler.

3.8.2.11 void glt_scheduler_set_data (GLT_sched sched, void * data)

Sets new data to a scheduler.

`glt_scheduler_set_data()` Sets data to a scheduler.

Parameters

in	<i>sched</i>	Handle of the target GLT_sched.
in	<i>data</i>	to be set.

3.9 Key functions

Functions

- void `glt_key_create` (void(*destructor)(void *value), `GLT_key` *newkey)
Creates a key.
- void `glt_key_free` (`GLT_key` *key)
Destroys a key.
- void `glt_key_set` (`GLT_key` key, void *value)
Sets new value to a key.
- void `glt_key_get` (`GLT_key` key, void **value)
Gets value from a key.

3.9.1 Detailed Description

These functions manage the GLT keys for the ULTs.

3.9.2 Function Documentation

3.9.2.1 void `glt_key_create` (void(*) (void *value) *destructor*, `GLT_key` * *newkey*)

Creates a key.

`glt_key_create()` creates a key.

Parameters

in	<i>destructor</i>	Handle to newly created <code>GLT_ult</code> .
out	<i>newkey</i>	Handle to newly created <code>GLT_key</code> .

3.9.2.2 void `glt_key_free` (`GLT_key` * *key*)

Destroys a key.

`glt_key_free()` destroys a key for ULTs.

Parameters

in	<i>key</i>	Handle to the target <code>GLT_key</code> .
----	------------	---------------------------------------------

3.9.2.3 void `glt_key_get` (`GLT_key` *key*, void ** *value*)

Gets value from a key.

`glt_key_get()` Gets value from a key.

Parameters

in	<i>key</i>	Handle of the target <code>GLT_key</code> .
out	<i>value</i>	obtained value.

3.9.2.4 void `glt_key_set` (`GLT_key` *key*, void * *value*)

Sets new value to a key.

`glt_key_set()` Sets value to a key.

Parameters

in	<i>key</i>	Handle of the target GLT_key.
in	<i>value</i>	to be set.

3.10 GLT object list

Variables

- [GLT_ult](#)
The user level thread abstraction.
- [GLT_tasklet](#)
The tasklet abstraction.
- [GLT_thread](#)
The thread abstraction.
- [GLT_mutex](#)
The mutex abstraction.
- [GLT_barrier](#)
The barrier abstraction.
- [GLT_cond](#)
The condition abstraction.
- [GLT_timer](#)
The timer abstraction.
- [GLT_bool](#)
The boolean abstraction.
- [GLT_thread_id](#)
The thread id abstraction.
- [GLT_ult_id](#)
The ult id abstraction.
- [GLT_key](#)
The key abstraction.
- [GLT_sched](#)
The scheduler abstraction.
- [GLT_sched_config](#)
The scheduler configuration abstraction.
- [GLT_sched_def](#)
The scheduler definition abstraction.
- [GLT_sched_predef](#)
The scheduler predefinition abstraction.

3.10.1 Detailed Description

Index

Barrier functions, [7](#)

- [glt_barrier_create](#), [7](#)
- [glt_barrier_free](#), [7](#)
- [glt_barrier_wait](#), [7](#)

Condition functions, [8](#)

- [glt_cond_broadcast](#), [8](#)
- [glt_cond_create](#), [8](#)
- [glt_cond_free](#), [8](#)
- [glt_cond_signal](#), [8](#)
- [glt_cond_wait](#), [9](#)

GLT object list, [27](#)

- [glt_barrier_create](#)
 - Barrier functions, [7](#)
- [glt_barrier_free](#)
 - Barrier functions, [7](#)
- [glt_barrier_wait](#)
 - Barrier functions, [7](#)
- [glt_cond_broadcast](#)
 - Condition functions, [8](#)
- [glt_cond_create](#)
 - Condition functions, [8](#)
- [glt_cond_free](#)
 - Condition functions, [8](#)
- [glt_cond_signal](#)
 - Condition functions, [8](#)
- [glt_cond_wait](#)
 - Condition functions, [9](#)
- [glt_end](#)
 - Library functions, [5](#)
- [glt_finalize](#)
 - Library functions, [5](#)
- [glt_get_num_threads](#)
 - Util functions, [20](#)
- [glt_get_thread_num](#)
 - Util functions, [20](#)
- [glt_get_wtime](#)
 - Timer functions, [18](#)
- [glt_init](#)
 - Library functions, [5](#)
- [glt_key_create](#)
 - Key functions, [25](#)
- [glt_key_free](#)
 - Key functions, [25](#)
- [glt_key_get](#)
 - Key functions, [25](#)
- [glt_key_set](#)
 - Key functions, [25](#)
- [glt_mutex_create](#)

Mutex functions, [10](#)

- [glt_mutex_free](#)
 - Mutex functions, [10](#)
- [glt_mutex_lock](#)
 - Mutex functions, [10](#)
- [glt_mutex_trylock](#)
 - Mutex functions, [10](#)
- [glt_mutex_unlock](#)
 - Mutex functions, [11](#)
- [glt_scheduler_create_basic](#)
 - Scheduler functions, [21](#)
- [glt_scheduler_config_free](#)
 - Scheduler functions, [22](#)
- [glt_scheduler_create](#)
 - Scheduler functions, [22](#)
- [glt_scheduler_exit](#)
 - Scheduler functions, [22](#)
- [glt_scheduler_finish](#)
 - Scheduler functions, [22](#)
- [glt_scheduler_free](#)
 - Scheduler functions, [22](#)
- [glt_scheduler_get_data](#)
 - Scheduler functions, [23](#)
- [glt_scheduler_get_size](#)
 - Scheduler functions, [23](#)
- [glt_scheduler_get_total_size](#)
 - Scheduler functions, [23](#)
- [glt_scheduler_has_to_stop](#)
 - Scheduler functions, [23](#)
- [glt_scheduler_set_data](#)
 - Scheduler functions, [23](#)
- [glt_start](#)
 - Library functions, [6](#)
- [glt_tasklet_cancel](#)
 - Work-units functions, [12](#)
- [glt_tasklet_create](#)
 - Work-units functions, [13](#)
- [glt_tasklet_create_to](#)
 - Work-units functions, [13](#)
- [glt_tasklet_join](#)
 - Work-units functions, [13](#)
- [glt_tasklet_malloc](#)
 - Work-units functions, [13](#)
- [glt_tasklet_self](#)
 - Work-units functions, [13](#)
- [glt_timer_create](#)
 - Timer functions, [18](#)
- [glt_timer_free](#)
 - Timer functions, [18](#)

- glt_timer_get_secs
 - Timer functions, [19](#)
- glt_timer_start
 - Timer functions, [19](#)
- glt_timer_stop
 - Timer functions, [19](#)
- glt_ult_cancel
 - Work-units functions, [15](#)
- glt_ult_create
 - Work-units functions, [15](#)
- glt_ult_create_to
 - Work-units functions, [15](#)
- glt_ult_exit
 - Work-units functions, [15](#)
- glt_ult_get_id
 - Work-units functions, [15](#)
- glt_ult_join
 - Work-units functions, [16](#)
- glt_ult_malloc
 - Work-units functions, [16](#)
- glt_ult_migrate_self_to
 - Work-units functions, [16](#)
- glt_ult_self
 - Work-units functions, [16](#)
- glt_workunit_get_thread_id
 - Work-units functions, [16](#)
- glt_yield
 - Work-units functions, [17](#)
- glt_yield_to
 - Work-units functions, [17](#)
- Key functions, [25](#)
 - glt_key_create, [25](#)
 - glt_key_free, [25](#)
 - glt_key_get, [25](#)
 - glt_key_set, [25](#)
- Library functions, [5](#)
 - glt_end, [5](#)
 - glt_finalize, [5](#)
 - glt_init, [5](#)
 - glt_start, [6](#)
- Mutex functions, [10](#)
 - glt_mutex_create, [10](#)
 - glt_mutex_free, [10](#)
 - glt_mutex_lock, [10](#)
 - glt_mutex_trylock, [10](#)
 - glt_mutex_unlock, [11](#)
- Scheduler functions, [21](#)
 - glt_scheduler_create_basic, [21](#)
 - glt_scheduler_config_free, [22](#)
 - glt_scheduler_create, [22](#)
 - glt_scheduler_exit, [22](#)
 - glt_scheduler_finish, [22](#)
 - glt_scheduler_free, [22](#)
 - glt_scheduler_get_data, [23](#)
 - glt_scheduler_get_size, [23](#)
 - glt_scheduler_get_total_size, [23](#)
 - glt_scheduler_has_to_stop, [23](#)
 - glt_scheduler_set_data, [23](#)
- Timer functions, [18](#)
 - glt_get_wtime, [18](#)
 - glt_timer_create, [18](#)
 - glt_timer_free, [18](#)
 - glt_timer_get_secs, [19](#)
 - glt_timer_start, [19](#)
 - glt_timer_stop, [19](#)
- Util functions, [20](#)
 - glt_get_num_threads, [20](#)
 - glt_get_thread_num, [20](#)
- Work-units functions, [12](#)
 - glt_tasklet_cancel, [12](#)
 - glt_tasklet_create, [13](#)
 - glt_tasklet_create_to, [13](#)
 - glt_tasklet_join, [13](#)
 - glt_tasklet_malloc, [13](#)
 - glt_tasklet_self, [13](#)
 - glt_ult_cancel, [15](#)
 - glt_ult_create, [15](#)
 - glt_ult_create_to, [15](#)
 - glt_ult_exit, [15](#)
 - glt_ult_get_id, [15](#)
 - glt_ult_join, [16](#)
 - glt_ult_malloc, [16](#)
 - glt_ult_migrate_self_to, [16](#)
 - glt_ult_self, [16](#)
 - glt_workunit_get_thread_id, [16](#)
 - glt_yield, [17](#)
 - glt_yield_to, [17](#)