

torque Reference Manual

2.3.6

Generated by Doxygen 1.4.7

Fri Feb 20 13:11:29 2009

Contents

1 torque Module Index	1
2 torque Data Structure Index	1
3 torque File Index	1
4 torque Module Documentation	2
5 torque Data Structure Documentation	5
6 torque File Documentation	8

1 torque Module Index

1.1 torque Modules

Here is a list of all modules:

DRMAA interface.	2
Vector iteration functions.	4
Job template operations.	2
Session managing function.	4
Remembering submitted job in session.	4

2 torque Data Structure Index

2.1 torque Data Structures

Here are the data structures with brief descriptions:

drmaa_job_iter_s (Iterates over submitted jobs set)	5
drmaa_job_s (Job data stored for each submitted job)	6
drmaa_job_template_s (Job template data)	6
drmaa_session_s (DRMAA session data)	7

3 torque File Index

3.1 torque File List

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

src/attrib.h	??
src/compat.h (System compatibility functions)	8
src/drmmaa.h (DRMAA library for Torque/PBS)	8
src/drmmaa_impl.h	??
src/error.h (Rasing errors)	14
src/jobs.h (Remembering submitted job in session)	14
src/lookup3.h (32bit hash function implementation)	15

4 torque Module Documentation

4.1 Job template operations.

This template is used to describe the job to be submitted. This description is accomplished by setting the desired scalar and vector attributes to their appropriate values. This template is then used in the job submission process.

4.2 DRMAA interface.

Modules

- [Vector iteration functions.](#)

The `drmaa_get_next_X()` functions SHALL store up to value_len bytes of the next attribute name / attribute value / job identifier from the values opaque string vector in the value buffer.

- [Job template operations.](#)

The function `drmaa_allocate_job_template()` SHALL allocate a new job template, returned in jt.

Functions

- int [drmaa_init](#) (const char *contact, char *error_diagnosis, size_t error_diag_len)

The `drmaa_init()` function SHALL initialize DRMAA library and create a new DRMAA session, using the contact parameter, if provided, to determine to which DRMS to connect.

- int [drmaa_exit](#) (char *error_diagnosis, size_t error_diag_len)

The `drmaa_exit()` function SHALL disengage from DRMAA library and allow the DRMAA library to perform any necessary internal cleanup.

- `int drmaa_get_attribute (drmaa_job_template_t *jt, const char *name, char *value, size_t value_len, char *error_diagnosis, size_t error_diag_len)`

The function `drmaa_get_attribute()` SHALL fill the value buffer with up to `value_len` characters of the scalar attribute, `name`'s, value in the given job template.

- `int drmaa_set_vector_attribute (drmaa_job_template_t *jt, const char *name, const char *value[], char *error_diagnosis, size_t error_diag_len)`

The function `drmaa_set_vector_attribute()` SHALL set the vector attribute, `name`, in the job template, `jt`, to the value(s), `value`.

- `int drmaa_get_vector_attribute (drmaa_job_template_t *jt, const char *name, drmaa_attr_values_t **values, char *error_diagnosis, size_t error_diag_len)`

The function `drmaa_get_vector_attribute()` SHALL store in `values` an opaque values string vector containing the values of the vector attribute, `name`'s, value in the given job template.

- `int drmaa_get_attribute_names (drmaa_attr_names_t **values, char *error_diagnosis, size_t error_diag_len)`

The function `drmaa_get_attribute_names()` SHALL return the set of supported scalar attribute names in an opaque names string vector stored in `values`.

- `int drmaa_get_vector_attribute_names (drmaa_attr_names_t **values, char *error_diagnosis, size_t error_diag_len)`

The function `drmaa_get_vector_attribute_names()` SHALL return the set of supported vector attribute names in an opaque names string vector stored in `values`.

4.2.1 Function Documentation

4.2.1.1 `int drmaa_init (const char * contact, char * error_diagnosis, size_t error_diag_len)`

The `drmaa_init()` function SHALL initialize DRMAA library and create a new DRMAA session, using the `contact` parameter, if provided, to determine to which DRMS to connect.

This function MUST be called before any other DRMAA function, except for `drmaa_get_DRM_system()`, `drmaa_get_DRMAA_implementation()`, `drmaa_get_contact()`, and `drmaa_strerror()`. If `contact` is NULL, the default DRM system SHALL be used, provided there is only one DRMAA implementation in the provided binary module. When there is more than one DRMAA implementation in the binary module, `drmaa_init()` SHALL return the `DRMAA_ERRNO_NO_DEFAULT_CONTACT_STRING_SELECTED` error code. The `drmaa_init()` function SHOULD be called by only one of the threads. The main thread is RECOMMENDED. A call by another thread SHALL return the `DRMAA_ERRNO_ALREADY_ACTIVE_SESSION` error code.

4.2.1.2 `int drmaa_exit (char * error_diagnosis, size_t error_diag_len)`

The `drmaa_exit()` function SHALL disengage from DRMAA library and allow the DRMAA library to perform any necessary internal cleanup.

This routine SHALL end the current DRMAA session but SHALL NOT affect any jobs (e.g. queued and running jobs SHALL remain queued and running). `drmaa_exit()` SHOULD be called by only one of the threads. The first call to call `drmaa_exit()` by a thread will operate normally. All other calls from the same and other threads SHALL fail, returning a `DRMAA_ERRNO_NO_ACTIVE_SESSION` error code.

4.2.1.3 `int drmaa_set_vector_attribute (drmaa_job_template_t *jt, const char *name, const char *value[], char *error_diagnosis, size_t error_diag_len)`

The function `drmaa_set_vector_attribute()` SHALL set the vector attribute, *name*, in the job template, *jt*, to the value(s), *value*.

The DRMAA implementation MUST accept value values that are arrays of one or more strings terminated by a NULL entry.

4.2.1.4 `int drmaa_get_attribute_names (drmaa_attr_names_t **values, char *error_diagnosis, size_t error_diag_len)`

The function `drmaa_get_attribute_names()` SHALL return the set of supported scalar attribute names in an opaque names string vector stored in *values*.

This vector SHALL include all required scalar attributes, all supported optional scalar attributes, all DRM-specific scalar attributes, and no unsupported optional attributes.

4.2.1.5 `int drmaa_get_vector_attribute_names (drmaa_attr_names_t **values, char *error_diagnosis, size_t error_diag_len)`

The function `drmaa_get_vector_attribute_names()` SHALL return the set of supported vector attribute names in an opaque names string vector stored in *values*.

This vector SHALL include all required vector attributes, all supported optional vector attributes, all DRM-specific vector attributes, and no unsupported optional attributes.

4.3 Vector iteration functions.

The opaque string vector's internal iterator SHALL then be moved forward to the next entry. If there are no more values those functions return DRMAA_ERRNO_INVALID_ARGUMENT (but this is outside DRMAA specification).

The `drmaa_get_num_X()` functions SHALL store the number of elements in the space provided by *size*.

The `drmaa_release_X()` functions free the memory used by the *values* opaque string vector. All memory used by strings contained therein is also freed.

4.4 Session managing function.

Functions

- `int drmaa_create (drmaa_session_t **pc, const char *contact, char *errmsg, size_t errlen)`
Creates DRMAA session and opens connection with DRM.
- `int drmaa_destroy (drmaa_session_t *c, char *errmsg, size_t errlen)`
Closes connection with DRM (if any) and destroys DRMAA session data.

4.5 Remembering submitted job in session.

Data Structures

- struct `drmaa_job_s`

Job data stored for each submitted job.

- struct `drmaa_job_iter_s`

Iterates over submitted jobs set.

Functions

- void `drmaa_get_job_list_iter` (`drmaa_session_t` *session, `drmaa_job_iter_t` *iter)
Returns iterator to jobs held in DRMAA session.
- `drmaa_job_t` * `drmaa_get_next_job` (`drmaa_job_iter_t` *iter)
Returns next job identifier from set or NULL if set finished.
- void `drmaa_add_job` (`drmaa_session_t` *c, `drmaa_job_t` *job)
Adds job identifier to session.
- bool `drmaa_find_job` (`drmaa_session_t` *c, const char *jobid, `drmaa_job_t` *found, unsigned flags)
Checks if job with given identifier exist in hash table and optionally removes it.

4.5.1 Function Documentation

4.5.1.1 void drmaa_get_job_list_iter (drmaa_session_t * session, drmaa_job_iter_t * iter)

Returns iterator to jobs held in DRMAA session.

Caller thread should have `drmaa_session_s::jobs_mutex` acquired iterator remains valid until job list is modified (or lock is released).

4.5.1.2 void drmaa_add_job (drmaa_session_t * c, drmaa_job_t * job)

Adds job identifier to session.

Parameters:

c DRMAA session.

job Malloced `drmaa_job_t` structure with filled jobid field (also malloced).

4.5.1.3 bool drmaa_find_job (drmaa_session_t * c, const char * jobid, drmaa_job_t * found, unsigned flags)

Checks if job with given identifier exist in hash table and optionally removes it.

Parameters:

c Opened DRMAA session.

jobid Job identifier.

found If not NULL and job was found job session data will be stored here.

flags Information to store into session. If DRMAA_JOB_DISPOSE bit is set session data will be removed.

See also:

`job_flag_t`

5 torque Data Structure Documentation

5.1 drmaa_job_iter_s Struct Reference

```
#include <jobs.h>
```

5.1.1 Detailed Description

Iterates over submitted jobs set.

Data Fields

- unsigned [hash](#)
Hash value of job identifier.

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

- [src/jobs.h](#)

5.2 drmaa_job_s Struct Reference

```
#include <jobs.h>
```

5.2.1 Detailed Description

Job data stored for each submitted job.

It is freed when job terminates and it's status is disposed by [drmaa_wait\(\)](#) or [drmaa_synchronize\(\)](#).

Data Fields

- [drmaa_job_t * next](#)
Next job in list or NULL.
- [char * jobid](#)
Job identifier from DRM.
- [int time_label](#)
Job submission timestamp increased in DRMAA session with each submitted job.
- [bool terminated](#)
Whether we know that job terminated and its status is waiting to rip.

- bool [suspended](#)

Whether job was suspended within session by [drmaa_control\(\)](#).

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

- [src/jobs.h](#)

5.3 drmaa_job_template_s Struct Reference

```
#include <drmaa_impl.h>
```

5.3.1 Detailed Description

Job template data.

Data Fields

- [drmaa_session_t](#) * [session](#)
DRMAA session in which job template was created.
- [drmaa_job_template_t](#) * [prev](#)
Previous job template in list.
- [drmaa_job_template_t](#) * [next](#)
Next job template in list.
- void ** [attrib](#)
Table of DRMAA attributes.
- [pthread_mutex_t](#) [mutex](#)
Mutex for accessing job attributes.

5.3.2 Field Documentation

5.3.2.1 void** [drmaa_job_template_s::attrib](#)

Table of DRMAA attributes.

It is filled with N_DRMAA_ATTRIBS values which are either NULL (attribute not set) or string (scalar attribute) or NULL terminated array of strings (vector attribute).

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

- [src/drmaa_impl.h](#)

5.4 drmaa_session_s Struct Reference

```
#include <drmaa_impl.h>
```


5.4.1 Detailed Description

DRMAA session data.

Data Fields

- int `pbs_conn`
PBS connection (or -1).
- char * `contact`
Contact to PBS server – ‘host[:port]’.
- `drmaa_job_template_t` * `jt_list`
Cyclic list (with sentinel) of job templates created in this DRMAA session.
- `drmaa_job_t` ** `job_hashtab`
Hash table of jobs which have to be remembered in DRMAA session (was submitted in this session and its status was not removed).
- int `next_time_label`
Will be assigned to next submitted job.
- pthread_mutex_t `conn_mutex`
Mutex for PBS connection.
- pthread_mutex_t `jobs_mutex`
Mutex for `jt_list`, `job_list` and `next_time_label`.

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

- `src/drmaa_impl.h`

6 torque File Documentation

6.1 `src/compat.h` File Reference

6.1.1 Detailed Description

System compatibility functions.

```
#include <stddef.h>
```

```
#include <stdarg.h>
```

6.2 `src/drmaa.h` File Reference

6.2.1 Detailed Description

DRMAA library for Torque/PBS.

Author:

Lukasz Ciesnik <lukasz.ciesnik@gmail.com>

Copyright (C) 2006 Poznan Supercomputing and Networking Center DSP team <dsp-devel@hedera.man.poznan.pl>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Documentation taken from:

Distributed Resource Management Application API C Bindings v1.0

Copyright (C) Global Grid Forum (2003). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the GGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the GGF Document process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the GGF or its successors or assigns. This document and the information contained herein is provided on an "AS IS" basis and THE GLOBAL GRID FORUM DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

```
#include <stddef.h>
```

```
#include <stdio.h>
```

Functions

- int [drmaa_init](#) (const char *contact, char *error_diagnosis, size_t error_diag_len)

The [drmaa_init\(\)](#) function SHALL initialize DRMAA library and create a new DRMAA session, using the contact parameter, if provided, to determine to which DRMS to connect.

- int [drmaa_exit](#) (char *error_diagnosis, size_t error_diag_len)

The [drmaa_exit\(\)](#) function SHALL disengage from DRMAA library and allow the DRMAA library to perform any necessary internal cleanup.

- int [drmaa_get_attribute](#) (drmaa_job_template_t *jt, const char *name, char *value, size_t value_len, char *error_diagnosis, size_t error_diag_len)

The function [drmaa_get_attribute\(\)](#) SHALL fill the value buffer with up to value_len characters of the scalar attribute, name's, value in the given job template.

- `int drmaa_set_vector_attribute (drmaa_job_template_t *jt, const char *name, const char *value[], char *error_diagnosis, size_t error_diag_len)`
The function `drmaa_set_vector_attribute()` SHALL set the vector attribute, name, in the job template, jt, to the value(s), value.
- `int drmaa_get_vector_attribute (drmaa_job_template_t *jt, const char *name, drmaa_attr_values_t **values, char *error_diagnosis, size_t error_diag_len)`
The function `drmaa_get_vector_attribute()` SHALL store in values an opaque values string vector containing the values of the vector attribute, name's, value in the given job template.
- `int drmaa_get_attribute_names (drmaa_attr_names_t **values, char *error_diagnosis, size_t error_diag_len)`
The function `drmaa_get_attribute_names()` SHALL return the set of supported scalar attribute names in an opaque names string vector stored in values.
- `int drmaa_get_vector_attribute_names (drmaa_attr_names_t **values, char *error_diagnosis, size_t error_diag_len)`
The function `drmaa_get_vector_attribute_names()` SHALL return the set of supported vector attribute names in an opaque names string vector stored in values.
- `int drmaa_run_job (char *job_id, size_t job_id_len, const drmaa_job_template_t *jt, char *error_diagnosis, size_t error_diag_len)`
The `drmaa_run_job()` function submits a single job with the attributes defined in the job template, jt.
- `int drmaa_run_bulk_jobs (drmaa_job_ids_t **jobids, const drmaa_job_template_t *jt, int start, int end, int incr, char *error_diagnosis, size_t error_diag_len)`
The `drmaa_run_bulk_jobs()` function submits a set of parametric jobs which can be run concurrently.
- `int drmaa_control (const char *job_id, int action, char *error_diagnosis, size_t error_diag_len)`
The `drmaa_control()` function SHALL enact the action indicated by action on the job specified by the job identifier, jobid.
- `int drmaa_job_ps (const char *job_id, int *remote_ps, char *error_diagnosis, size_t error_diag_len)`
The `drmaa_job_ps()` function SHALL store in remote_ps the program status of the job identified by job_id.
- `int drmaa_synchronize (const char *job_ids[], signed long timeout, int dispose, char *error_diagnosis, size_t error_diag_len)`
The `drmaa_synchronize()` function SHALL cause the calling thread to block until all jobs specified by job_ids have finished execution.
- `int drmaa_wait (const char *job_id, char *job_id_out, size_t job_id_out_len, int *stat, signed long timeout, drmaa_attr_values_t **usage, char *error_diagnosis, size_t error_diag_len)`
The `drmaa_wait()` function SHALL wait for a job identified by job_id to finish execution or fail.
- `const char * drmaa_strerror (int drmaa_errno)`
The `drmaa_strerror()` function SHALL return the error string describing the DRMAA error number drmaa_errno.
- `int drmaa_get_contact (char *contact, size_t contact_len, char *error_diagnosis, size_t error_diag_len)`

The `drmaa_get_contact()` function, if called before `drmaa_init()`, SHALL return a string containing a comma-delimited list of default DRMAA implementation contacts strings, one per DRM implementation provided.

- int `drmaa_version` (unsigned int *major, unsigned int *minor, char *error_diagnosis, size_t error_diag_len)

The `drmaa_version()` function SHALL set major and minor to the major and minor versions of the DRMAA C binding specification implemented by the DRMAA implementation.

- int `drmaa_get_DRM_system` (char *drm_system, size_t drm_system_len, char *error_diagnosis, size_t error_diag_len)

The `drmaa_get_DRM_system()` function, if called before `drmaa_init()`, SHALL return a string containing a comma-delimited list of DRM system identifiers, one per DRM system implementation provided.

- int `drmaa_get_DRMAA_implementation` (char *drmaa_impl, size_t drmaa_impl_len, char *error_diagnosis, size_t error_diag_len)

The `drmaa_get_DRMAA_implementation()` function, if called before `drmaa_init()`, SHALL return a string containing a comma-delimited list of DRMAA implementations, one per DRMAA implementation provided.

- void `drmaa_set_logging_output` (FILE *file)

Specify place where goes log messages from library when they were enabled at configure time.

6.2.2 Function Documentation

6.2.2.1 int `drmaa_run_job` (char *job_id, size_t job_id_len, const `drmaa_job_template_t` *jt, char *error_diagnosis, size_t error_diag_len)

The `drmaa_run_job()` function submits a single job with the attributes defined in the job template, `jt`.

Upon success, up to `job_id_len` characters of the submitted job's job identifier are stored in the buffer, `job_id`.

6.2.2.2 int `drmaa_run_bulk_jobs` (`drmaa_job_ids_t` **jobids, const `drmaa_job_template_t` *jt, int start, int end, int incr, char *error_diagnosis, size_t error_diag_len)

The `drmaa_run_bulk_jobs()` function submits a set of parametric jobs which can be run concurrently.

The attributes defined in the job template, `jt` are used for every parametric job in the set. Each job in the set is identical except for it's index. The first parametric job has an index equal to `start`. The next job has an index equal to `start + incr`, and so on. The last job has an index equal to `start + n * incr`, where `n` is equal to $(end - start) / incr$. Note that the value of the last job's index may not be equal to `end` if the difference between `start` and `end` is not evenly divisible by `incr`. The smallest valid value for `start` is 1. The largest valid value for `end` is 2147483647 ($2^{31} - 1$). The `start` value must be less than or equal to the `end` value, and only positive index numbers are allowed. The index number can be determined by the job in an implementation specific fashion. On success, an opaque job id string vector containing job identifiers for all submitted jobs SHALL be returned into `job_ids`. The job identifiers in the opaque job id string vector can be extracted using the `drmaa_get_next_job_id()` function. The caller is responsible for releasing the opaque job id string vector returned into `job_ids` using the `drmaa_release_job_ids()` function.

6.2.2.3 int `drmaa_control` (const char *job_id, int action, char *error_diagnosis, size_t error_diag_len)

The `drmaa_control()` function SHALL enact the action indicated by *action* on the job specified by the job identifier, *jobid*.

The action parameter's value may be one of the following:

- `DRMAA_CONTROL_SUSPEND`
- `DRMAA_CONTROL_RESUME`
- `DRMAA_CONTROL_HOLD`
- `DRMAA_CONTROL_RELEASE`
- `DRMAA_CONTROL_TERMINATE` The `drmaa_control()` function SHALL return after the DRM system has acknowledged the command, not necessarily after the desired action has been performed. If *jobid* is `DRMAA_JOB_IDS_SESSION_ALL`, this function SHALL perform the specified action on all jobs submitted during this session as of this function is called.

6.2.2.4 `int drmaa_job_ps (const char * job_id, int * remote_ps, char * error_diagnosis, size_t error_diag_len)`

The `drmaa_job_ps()` function SHALL store in *remote_ps* the program status of the job identified by *job_id*.

The possible values of a program's status are:

- `DRMAA_PS_UNDETERMINED`
- `DRMAA_PS_QUEUED_ACTIVE`
- `DRMAA_PS_SYSTEM_ON_HOLD`
- `DRMAA_PS_USER_ON_HOLD`
- `DRMAA_PS_USER_SYSTEM_ON_HOLD`
- `DRMAA_PS_RUNNING`
- `DRMAA_PS_SYSTEM_SUSPENDED`
- `DRMAA_PS_USER_SUSPENDED`
- `DRMAA_PS_DONE`
- `DRMAA_PS_FAILED`

Terminated jobs have a status of `DRMAA_PS_FAILED`.

6.2.2.5 `int drmaa_synchronize (const char * job_ids[], signed long timeout, int dispose, char * error_diagnosis, size_t error_diag_len)`

The `drmaa_synchronize()` function SHALL cause the calling thread to block until all jobs specified by *job_ids* have finished execution.

If *job_ids* contains `DRMAA_JOB_IDS_SESSION_ALL`, then this function SHALL wait for all jobs submitted during this DRMAA session as of the point in time when `drmaa_synchronize()` is called. To avoid thread race conditions in multithreaded applications, the DRMAA implementation user should explicitly synchronize this call with any other job submission calls or control calls that may change the number of remote jobs.

The *timeout* parameter value indicates how many seconds to remain blocked in this call waiting for results to become available, before returning with a `DRMAA_ERRNO_EXIT_TIMEOUT` error code. The value, `DRMAA_TIMEOUT_WAIT_FOREVER`, MAY be specified to wait indefinitely for a result. The value, `DRMAA_TIMEOUT_NO_WAIT`, MAY be specified to return immediately with a `DRMAA_ERRNO_EXIT_TIMEOUT` error code if no result is available. If the call exits before the timeout has elapsed, all the jobs have been waited on or there was an interrupt. The caller should check system time before and after this call in order to be sure of how much time has passed. The *dispose* parameter specifies how to treat the reaping of the remote job's internal data record, which includes a record of the job's consumption of system resources during its execution and other statistical information. If the *dispose* parameter's value is 1, the DRMAA implementation SHALL dispose of the job's data record at the end of the `drmaa_synchronize()` call. If the *dispose* parameter's value is 0, the data record SHALL be left for future access via the `drmaa_wait()` method.

6.2.2.6 `int drmaa_wait (const char * job_id, char * job_id_out, size_t job_id_out_len, int * stat, signed long timeout, drmaa_attr_values_t ** rusage, char * error_diagnosis, size_t error_diag_len)`

The `drmaa_wait()` function SHALL wait for a job identified by *job_id* to finish execution or fail.

If the special string, `JOB_IDS_SESSION_ANY`, is provided as the *job_id*, this function will wait for any job from the session to finish execution or fail. In this case, any job for which exit status information is available will satisfy the requirement, including jobs which previously finished but have never been the subject of a `drmaa_wait()` call. This routine is modeled on the `wait3` POSIX routine.

The *timeout* parameter value indicates how many seconds to remain blocked in this call waiting for a result, before returning with a `DRMAA_ERRNO_EXIT_TIMEOUT` error code. The value, `DRMAA_TIMEOUT_WAIT_FOREVER`, MAY be specified to wait indefinitely for a result. The value, `DRMAA_TIMEOUT_NO_WAIT`, MAY be specified to return immediately with a `DRMAA_ERRNO_EXIT_TIMEOUT` error code if no result is available. If the call exits before the timeout has elapsed, the job has been successfully waited on or there was an interrupt. The caller should check system time before and after this call in order to be sure of how much time has passed.

Upon success, `drmaa_wait()` fills *job_id_out* with up to *job_id_out_len* characters of the waited job's id, *stat* with the a code that includes information about the conditions under which the job terminated, and *rusage* with an array of `<name>=<value>` strings that describe the amount of resources consumed by the job and are implementation defined. The *stat* parameter is further described below. The *rusage* parameter's values may be accessed via `drmaa_get_next_attr_value()`.

The `drmaa_wait()` function reaps job data records on a successful call, so any subsequent calls to `drmaa_wait()` will fail, returning a `DRMAA_ERRNO_INVALID_JOB` error code, meaning that the job's data record has already been reaped. This error code is the same as if the job were unknown. If `drmaa_wait()` exists due to a timeout, `DRMAA_ERRNO_EXIT_TIMEOUT` is returned and no *rusage* information is reaped. (The only case where `drmaa_wait()` can be successfully called on a single job more than once is when the previous call(s) to `drmaa_wait()` returned `DRMAA_ERRNO_EXIT_TIMEOUT`.)

The *stat* parameter, set by a successful call to `drmaa_wait()`, is used to retrieve further input about the exit condition of the waited job, identified by *job_id_out*, through the following functions: `drmaa_wifexited()`, `drmaa_wexitstatus()`, `drmaa_wifsignaled()`, `drmaa_wtermsig()`, `drmaa_wcoredump()` and `drmaa_wifaborted()`.

6.2.2.7 `int drmaa_get_contact (char * contact, size_t contact_len, char * error_diagnosis, size_t error_diag_len)`

The `drmaa_get_contact()` function, if called before `drmaa_init()`, SHALL return a string containing a comma-delimited list of default DRMAA implementation contacts strings, one per DRM implementation provided.

If called after [drmaa_init\(\)](#), [drmaa_get_contacts\(\)](#) SHALL return the contact string for the DRM system for which the library has been initialized.

6.2.2.8 int drmaa_get_DRM_system (char * *drm_system*, size_t *drm_system_len*, char * *error_diagnosis*, size_t *error_diag_len*)

The [drmaa_get_DRM_system\(\)](#) function, if called before [drmaa_init\(\)](#), SHALL return a string containing a comma-delimited list of DRM system identifiers, one per DRM system implementation provided.

If called after [drmaa_init\(\)](#), [drmaa_get_DRM_system\(\)](#) SHALL return the selected DRM system.

6.2.2.9 int drmaa_get_DRMAA_implementation (char * *drmaa_impl*, size_t *drmaa_impl_len*, char * *error_diagnosis*, size_t *error_diag_len*)

The [drmaa_get_DRMAA_implementation\(\)](#) function, if called before [drmaa_init\(\)](#), SHALL return a string containing a comma-delimited list of DRMAA implementations, one per DRMAA implementation provided.

If called after [drmaa_init\(\)](#), [drmaa_get_DRMAA_implementation\(\)](#) SHALL return the selected DRMAA implementation.

6.2.2.10 void drmaa_set_logging_output (FILE * *file*)

Specify place where goes log messages from library when they were enabled at configure time.

By default they are written to standard error stream.

Parameters:

file File to write to.

6.3 src/error.h File Reference

6.3.1 Detailed Description

Raising errors.

```
#include <stddef.h>
```

Functions

- int [drmaa_get_errno_error](#) (char **error_diagnosis*, size_t *error_diag_len*)
Gets last system error message and returns its code.
- int [drmaa_get_pbs_error](#) (char **error_diagnosis*, size_t *error_diag_len*)
Retrieves last PBS error message.
- int [drmaa_map_pbs_error](#) (int *pbs_errcode*)
Maps PBS error code into DMRAA code.

6.4 src/jobs.h File Reference

6.4.1 Detailed Description

Remembering submitted job in session.

```
#include <drmaa_impl.h>
```

Data Structures

- struct [drmaa_job_s](#)
Job data stored for each submitted job.
- struct [drmaa_job_iter_s](#)
Iterates over submitted jobs set.

Functions

- void [drmaa_get_job_list_iter](#) ([drmaa_session_t](#) *session, [drmaa_job_iter_t](#) *iter)
Returns iterator to jobs held in DRMAA session.
- [drmaa_job_t](#) * [drmaa_get_next_job](#) ([drmaa_job_iter_t](#) *iter)
Returns next job identifier from set or NULL if set finished.
- void [drmaa_add_job](#) ([drmaa_session_t](#) *c, [drmaa_job_t](#) *job)
Adds job identifier to session.
- bool [drmaa_find_job](#) ([drmaa_session_t](#) *c, const char *jobid, [drmaa_job_t](#) *found, unsigned flags)
Checks if job with given identifier exist in hash table and optionally removes it.

6.5 src/lookup3.h File Reference

6.5.1 Detailed Description

32bit hash function implementation.

Taken from: <http://burtleburtle.net/bob/hash/>

```
#include <pbs_config.h>
```

Functions

- uint32_t [hashword](#) (const uint32_t *k, size_t length, uint32_t initval)
This works on all machines.
- uint32_t [hashlittle](#) (const void *key, size_t length, uint32_t initval)
[hashlittle\(\)](#) – hash a variable-length key into a 32-bit value
- uint32_t [hashbig](#) (const void *key, size_t length, uint32_t initval)

hashbig(): This is the same as hashword() on big-endian machines.

6.5.2 Function Documentation

6.5.2.1 uint32_t hashword (const uint32 * *k*, size_t *length*, uint32 *initval*)

This works on all machines.

To be useful, it requires – that the key be an array of uint32's, and – that all your machines have the same endianness, and – that the length be the number of uint32's in the key

The function `hashword()` is identical to `hashlittle()` on little-endian machines, and identical to `hashbig()` on big-endian machines, except that the length has to be measured in uint32s rather than in bytes. `hashlittle()` is more complicated than `hashword()` only because `hashlittle()` has to dance around fitting the key bytes into registers.

Parameters:

- k* the key, an array of uint32 values
- length* the length of the key, in uint32s
- initval* the previous hash, or an arbitrary value

6.5.2.2 uint32_t hashlittle (const void * *key*, size_t *length*, uint32 *initval*)

`hashlittle()` – hash a variable-length key into a 32-bit value

Parameters:

- key* the key (the unaligned variable-length array of bytes)
- length* the length of the key, counting by bytes
- initval* can be any 4-byte value

Returns a 32-bit value. Every bit of the key affects every bit of the return value. Two keys differing by one or two bits will have totally different hash values.

The best hash table sizes are powers of 2. There is no need to do mod a prime (mod is sooo slow!). If you need less than 32 bits, use a bitmask. For example, if you need only 10 bits, do `h = (h & hashmask(10));` In which case, the hash table should have `hashsize(10)` elements.

If you are hashing `n` strings (`uint8 **`)`k`, do it like this: for (`i=0`, `h=0`; `i<n`; `++i`) `h = hashlittle(k[i], len[i], h);`

By Bob Jenkins, 2006. `bob_jenkins@burtleburtle.net`. You may use this code any way you wish, private, educational, or commercial. It's free.

Use for hash table lookup, or anything where one collision in 2^{32} is acceptable. Do NOT use for cryptographic purposes.

6.5.2.3 uint32_t hashbig (const void * *key*, size_t *length*, uint32 *initval*)

`hashbig()`: This is the same as `hashword()` on big-endian machines.

It is different from `hashlittle()` on all machines. `hashbig()` takes advantage of big-endian byte ordering.

Index

- attrib
 - [drmaa_job_template_s](#), [7](#)
 - drmaa
 - [drmaa_exit](#), [3](#)
 - [drmaa_get_attribute_names](#), [3](#)
 - [drmaa_get_vector_attribute_names](#), [3](#)
 - [drmaa_init](#), [3](#)
 - [drmaa_set_vector_attribute](#), [3](#)
 - DRMAA interface., [2](#)
 - drmaa.h
 - [drmaa_control](#), [11](#)
 - [drmaa_get_contact](#), [13](#)
 - [drmaa_get_DRM_system](#), [13](#)
 - [drmaa_get_DRMAA_implementation](#), [13](#)
 - [drmaa_job_ps](#), [11](#)
 - [drmaa_run_bulk_jobs](#), [11](#)
 - [drmaa_run_job](#), [11](#)
 - [drmaa_set_logging_output](#), [14](#)
 - [drmaa_synchronize](#), [12](#)
 - [drmaa_wait](#), [12](#)
 - [drmaa_add_job](#)
 - [jobs](#), [5](#)
 - [drmaa_control](#)
 - [drmaa.h](#), [11](#)
 - [drmaa_exit](#)
 - [drmaa](#), [3](#)
 - [drmaa_find_job](#)
 - [jobs](#), [5](#)
 - [drmaa_get_attribute_names](#)
 - [drmaa](#), [3](#)
 - [drmaa_get_contact](#)
 - [drmaa.h](#), [13](#)
 - [drmaa_get_DRM_system](#)
 - [drmaa.h](#), [13](#)
 - [drmaa_get_DRMAA_implementation](#)
 - [drmaa.h](#), [13](#)
 - [drmaa_get_job_list_iter](#)
 - [jobs](#), [5](#)
 - [drmaa_get_vector_attribute_names](#)
 - [drmaa](#), [3](#)
 - [drmaa_init](#)
 - [drmaa](#), [3](#)
 - [drmaa_job_iter_s](#), [5](#)
 - [drmaa_job_ps](#)
 - [drmaa.h](#), [11](#)
 - [drmaa_job_s](#), [6](#)
 - [drmaa_job_template_s](#), [6](#)
 - [attrib](#), [7](#)
 - [drmaa_run_bulk_jobs](#)
 - [drmaa.h](#), [11](#)
 - [drmaa_run_job](#)
 - [drmaa.h](#), [11](#)
 - [drmaa_session_s](#), [7](#)
 - [drmaa_set_logging_output](#)
 - [drmaa.h](#), [14](#)
 - [drmaa_set_vector_attribute](#)
 - [drmaa](#), [3](#)
 - [drmaa_synchronize](#)
 - [drmaa.h](#), [12](#)
 - [drmaa_wait](#)
 - [drmaa.h](#), [12](#)
- [hashbig](#)
 - [lookup3.h](#), [16](#)
 - [hashlittle](#)
 - [lookup3.h](#), [16](#)
 - [hashword](#)
 - [lookup3.h](#), [15](#)
- [Job template operations.](#), [2](#)
 - [jobs](#)
 - [drmaa_add_job](#), [5](#)
 - [drmaa_find_job](#), [5](#)
 - [drmaa_get_job_list_iter](#), [5](#)
- [lookup3.h](#)
 - [hashbig](#), [16](#)
 - [hashlittle](#), [16](#)
 - [hashword](#), [15](#)
- [Remembering submitted job in session.](#), [4](#)
- [Session managing function.](#), [4](#)
 - [src/compat.h](#), [8](#)
 - [src/drmaa.h](#), [8](#)
 - [src/error.h](#), [14](#)
 - [src/jobs.h](#), [14](#)
 - [src/lookup3.h](#), [15](#)
- [Vector iteration functions.](#), [4](#)