GWD-R

Andreas Haas, Sun Microsystems (maintainer)
Roger Brobst, Cadence Design Systems
Andreas Haas*, Sun Microsystems
Nicholas Geib, Condor Group
Hrabri Rajic*, Intel Americas Inc.
John Tollefsrud[+], Sun Microsystems
*co-chairs
[+]founding co-chair

Distributed Resource Management
Application API (DRMAA) Working Group

May, 2004

Distributed Resource Management Application API C Bindings v0.95

Status of This Memo

This memo is a Global Grid Forum Grid Working Draft - Recommendations (GWD-R) in process, in general accordance with the provisions of Global Grid Forum Document GFD-C.1, the Global Grid Forum Documents and Recommendations: Process and Requirements, revised April 2002.

Abstract

This document describes the Distributed Resource Management Application API (DRMAA) C binding. The document is based on the implementations work of the DRMAA GWD-R document.

Table of Contents

1. Overview

This document lists a C language binding for the DRMAA interface. For information related to interface semantics, possible argument values, error conditions etc., consult chapter "3.2 DRMAA API" of the interface specification. The C header file below is complete only with regard to the information needed by a C compiler and linker.
This C binding may be used with C++ programs through use of the extern "C" { } wrapping technique, which is widely used to import C binding interfaces in C++ programs. An example is listed in the header file.

2. The C header file

The header file contains a C function prototype for each interface operation described in the DRMAA interface specification. The function names in this document are always identical with the names from the interface specification.

Function prototypes and opaque data types in the header file that do not have a counterpart in the interface specification are specific to the C language binding. The DRMAA interface makes frequent use of strings, string vectors as input and output arguments. Since C language does not have a “real” string data type, a few additional opaque data types and helper functions are used to handle output string vector arguments with the actual interface calls. To minimize the complexity that was added for the C language binding compared to the language independent specification, traditional C constructs such as "const char *" and "const char *job_ids[]" are used whenever possible. As a result not much has been added compared to the "3.2 DRMAA API" description.

```
SECTION 1. Compile time symbols

SECTION 1.1 Opaque data types

    The following four symbols SHALL be defined in a drmaa.h file in an
    opaque fashion such as

      typedef struct drmaa_job_template_s drmaa_job_template_t;
      typedef struct drmaa_attr_names_s drmaa_attr_names_t;
      typedef struct drmaa_attr_values_s drmaa_attr_values_t;
      typedef struct drmaa_job_ids_s  drmaa_job_ids_t;

    unintendend access to struct drmaa_*_s implementation dependent
    data members SHALL be precluded by leaving out struct drmaa_*_s
    from drmaa.h file. This ensures access to struct members can done
    only by means of the libraries access functions as described in
    sections after section 1.

SECTION 1.2 C preprocessor #defines for handling string output arguments

    Firstly, the following C preprocessor #define's SHALL be define
    in a drmaa.h file

         #define DRMAA_ATTR_BUFFER            1024
         #define DRMAA_CONTACT_BUFFER         1024
         #define DRMAA_DRM_SYSTEM_BUFFER      1024
         #define DRMAA_ERROR_STRING_BUFFER    1024
         #define DRMAA_JOBNAME_BUFFER         1024
```

```
#define DRMAA_SIGNAL_BUFFER           32
```

and could be used as global constants for buffer variable definition of type char *. The defined numbers above denote the recommended MINIMUM lengths for the content of the corresponding char * output variables. If buffers are passed that are smaller than the recommended minimum lengths the implementation MUST either truncate the string to be returned or indicate the DRMAA_ERRNO_INVALID_ARGUMENT error.

SECTION 1.3 C preprocessor #defines for control operations

Firstly these preprocessor #define's SHALL be defined in a drmaa.h file as follows

```
#define DRMAA_TIMEOUT_NO_WAIT          0
#define DRMAA_TIMEOUT_WAIT_FOREVER    -1
```

for convenience in programs that make use of drmaa_synchronize() or drmaa_wait().

Secondly these preprocessor #define's SHALL be defined in a drmaa.h file as follows

```
#define DRMAA_PS_UNDETERMINED          0x00

#define DRMAA_PS_QUEUED_ACTIVE         0x10
#define DRMAA_PS_SYSTEM_ON_HOLD        0x11
#define DRMAA_PS_USER_ON_HOLD          0x12
#define DRMAA_PS_USER_SYSTEM_ON_HOLD   0x13

#define DRMAA_PS_RUNNING               0x20
#define DRMAA_PS_SYSTEM_SUSPENDED      0x21
#define DRMAA_PS_USER_SUSPENDED        0x22
#define DRMAA_PS_USER_SYSTEM_SUSPENDED 0x23

#define DRMAA_PS_DONE                  0x30

#define DRMAA_PS_FAILED                0x40
```

for convenience in programs that make use of drmaa_job_ps().

Thirdly these preprocessor #define's SHALL be defined in a drmaa.h file as follows

```
#define DRMAA_CONTROL_SUSPEND          0
#define DRMAA_CONTROL_RESUME           1
#define DRMAA_CONTROL_HOLD             2
#define DRMAA_CONTROL_RELEASE          3
#define DRMAA_CONTROL_TERMINATE        4
```

for convenience in programs that make use of drmaa_control().

Forthly the following C preprocessor #define's SHALL be defined in a drmaa.h file as follows

```
#define DRMAA_JOB_IDS_SESSION_ALL     "DRMAA_JOB_IDS_SESSION_ANY"
```

```
#define DRMAA_JOB_IDS_SESSION_ANY      "DRMAA_JOB_IDS_SESSION_ALL"
```

for convenience in programs.

SECTION 1.4 C preprocessor #defines specifically for job template compilation

Firstly the following C preprocessor #define's SHALL be defined in a drmaa.h file as follows

```
#define DRMAA_SUBMISSION_STATE_ACTIVE  "drmaa_active"
#define DRMAA_SUBMISSION_STATE_HOLD    "drmaa_hold"

#define DRMAA_PLACEHOLDER_HD           "$drmaa_hd_ph$"
#define DRMAA_PLACEHOLDER_WD           "$drmaa_wd_ph$"

#define DRMAA_PLACEHOLDER_INCR         "$drmaa_incr_ph$"
```

for convenience in use of keywords within job template attribute values.

Secondly the following list of C preprocessor #define's SHALL be defined in a drmaa.h file as follows for convenience in use of job template attribute names

```
#define DRMAA_BLOCK_EMAIL              "drmaa_block_email"
#define DRMAA_DEADLINE_TIME            "drmaa_deadline_time"
#define DRMAA_DURATION_HLIMIT          "drmaa_durartion_hlimit"
#define DRMAA_DURATION_SLIMIT          "drmaa_durartion_slimit"
#define DRMAA_ERROR_PATH               "drmaa_error_path"
#define DRMAA_INPUT_PATH               "drmaa_input_path"
#define DRMAA_JOB_CATEGORY             "drmaa_job_category"
#define DRMAA_JOB_NAME                 "drmaa_job_name"
#define DRMAA_JOIN_FILES               "drmaa_join_files"
#define DRMAA_JS_STATE                 "drmaa_js_state"
#define DRMAA_NATIVE_SPECIFICATION
                                "drmaa_native_specification"
#define DRMAA_OUTPUT_PATH              "drmaa_output_path"
#define DRMAA_REMOTE_COMMAND           "drmaa_remote_command"
#define DRMAA_START_TIME               "drmaa_start_time"
#define DRMAA_TRANSFER_FILES
                                       "drmaa_transfer_files"
#define DRMAA_V_ARGV                   "drmaa_v_argv"
#define DRMAA_V_EMAIL                  "drmaa_v_email"
#define DRMAA_V_ENV                    "drmaa_v_env"
#define DRMAA_WCT_HLIMIT               "drmaa_wct_hlimit"
#define DRMAA_WCT_SLIMIT               "drmaa_wct_slimit"
#define DRMAA_WD                       "drmaa_wd"
```

SECTION 1.5 C preprocessor #defines used for DRMAA error codes

The following C preprocessor #define's SHALL be defined in a drmaa.h file as follows

```
#define DRMAA_ERRNO_SUCCESS                         0
#define DRMAA_ERRNO_INTERNAL_ERROR                  1
#define DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE       2
```

```
        #define DRMAA_ERRNO_AUTH_FAILURE                    3
        #define DRMAA_ERRNO_INVALID_ARGUMENT                4
        #define DRMAA_ERRNO_NO_ACTIVE_SESSION               5
        #define DRMAA_ERRNO_NO_MEMORY                       6
        #define DRMAA_ERRNO_INVALID_CONTACT_STRING          7
        #define DRMAA_ERRNO_DEFAULT_CONTACT_STRING_ERROR    8
        #define DRMAA_ERRNO_DRMS_INIT_FAILED                9
        #define DRMAA_ERRNO_ALREADY_ACTIVE_SESSION         10
        #define DRMAA_ERRNO_DRMS_EXIT_ERROR                11

        #define DRMAA_ERRNO_INVALID_ATTRIBUTE_FORMAT       12
        #define DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE        13
        #define DRMAA_ERRNO_CONFLICTING_ATTRIBUTE_VALUES   14

        #define DRMAA_ERRNO_TRY_LATER                      15
        #define DRMAA_ERRNO_DENIED_BY_DRM                  16

        #define DRMAA_ERRNO_INVALID_JOB                    17
        #define DRMAA_ERRNO_RESUME_INCONSISTENT_STATE      18
        #define DRMAA_ERRNO_SUSPEND_INCONSISTENT_STATE     19
        #define DRMAA_ERRNO_HOLD_INCONSISTENT_STATE        20
        #define DRMAA_ERRNO_RELEASE_INCONSISTENT_STATE     21
        #define DRMAA_ERRNO_EXIT_TIMEOUT                   22
        #define DRMAA_ERRNO_NO_RUSAGE                      23


/* SECTION 2. string list helper functions */
int drmaa_get_next_attr_name(drmaa_attr_names_t* values, char *value,
            size_t value_len);
int drmaa_get_next_attr_value(drmaa_attr_values_t* values, char *value,
            size_t value_len);
int drmaa_get_next_job_id(drmaa_job_ids_t* values, char *value, size_t
            value_len);
void drmaa_release_attr_names( drmaa_attr_names_t* values );
void drmaa_release_attr_values( drmaa_attr_values_t* values );
void drmaa_release_job_ids( drmaa_job_ids_t* values );

/* SECTION 3. session mgmt */
int drmaa_init(const char *contact, char *error_diagnosis, size_t
            error_diag_len);
int drmaa_exit(char *error_diagnosis, size_t error_diag_len);

/* SECTION 4. job template */
   NAME
      drmaa_allocate_job_template, drmaa_delete_job_template,
      drmaa_set_attribute, drmaa_get_attribute,
                  drmaa_set_vector_attribute,
      drmaa_get_vector_attribute, drmaa_get_attribute_names,
      drmaa_get_vector_attribute_names - methods to build a job
                                         template

   SYNOPSIS
      int drmaa_allocate_job_template(drmaa_job_template_t **jt, char
                  *error_diagnosis, size_t error_diag_len);

      int drmaa_delete_job_template(drmaa_job_template_t *jt, char
             *error_diagnosis, size_t error_diag_len);
```

```
int drmaa_set_attribute(drmaa_job_template_t *jt, const char
      *name, const char *value, char *error_diagnosis, size_t
            error_diag_len);

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);

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

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);

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

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

DESCRIPTION

The function drmaa_allocate_job_template() allocates a new job template, returned in jt. This template is used to describe the job to be submitted. This is accomplished by setting the desired scalar and vector attributes to their appropriate values. This template is then used in the job submission process.

The function drmaa_delete_job_template() deallocate the job template pointed to by jt.

The function drmaa_set_attribute() sets scalar attribute, name, to the value, value, in the job template, jt. Similarly, the function drmaa_set_vector_attribute() sets the vector attribute, name, to the value(s), value. Here, value must be an array of one or more strings terminated by the null string.

The function drmaa_get_attribute() fills value with up to value_len characters of the scalar attribute name's value in the given job template. drmaa_get_vector_attribute() performs the same for vector attributes.

The function drmaa_get_attribute_names() returns the set of supported scalar attribute names in values. drmaa_get_vector_attribute_names() performs the same for vector attributes.

RETURNS

All functions return DRMAA_ERRNO_SUCCESS on success.

ERRORS

If any error condition occurs, drmaa_allocate_job_template(), drmaa_deallocate_job_template(), drmaa_set_attribute(), drmaa_get_attribute() drmaa_set_vector_attribute(), drmaa_get_vector_attribute(), drmaa_get_attribute_names() and

drmaa_get_vector_attribute_names() shall provide up to
error_diag_len characters of error related diagnosis
information in the buffer error_diagnosis and return an
appropriate error code from those described below.

The drmaa_allocate_job_template() and drmaa_delete_job_template()
functions may return DRMAA_ERRNO_NO_MEMORY,
DRMAA_ERRNO_INTERNAL_ERROR or
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE.

The drmaa_set_attribute() may return DRMAA_ERRNO_NO_MEMORY,
DRMAA_ERRNO_INTERNAL_ERROR,
DRMAA_ERRNO_INVALID_ATTRIBUTE_FORMAT,
DRMAA_ERRNO_INVALID_ARGUMENT,
DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE or
DRMAA_ERRNO_CONFLICTING_ATTRIBUTE_VALUES.

The drmaa_set_vector_attribute() may return DRMAA_ERRNO_NO_MEMORY,
DRMAA_ERRNO_INTERNAL_ERROR,
DRMAA_ERRNO_INVALID_ATTRIBUTE_FORMAT,
DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE or
DRMAA_ERRNO_CONFLICTING_ATTRIBUTE_VALUES.

DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE
Could not contact DRM system for this request.

DRMAA_ERRNO_INTERNAL_ERROR
Unexpected or internal error.

DRMAA_ERRNO_INVALID_ARGUMENT
The input value for an argument is invalid.

DRMAA_ERRNO_CONFLICTING_ATTRIBUTE_VALUES
The value of this attribute is conflicting with a previously
set attribute.

DRMAA_ERRNO_INVALID_ATTRIBUTE_FORMAT
The format for the job attribute is invalid.

DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE
The value for the job attribute is invalid.

DRMAA_ERRNO_NO_MEMORY
The system is unable to allocate the required memory.

SEE ALSO
drmaa_run_job(), drmaa_get_next_attr_name(),
drmaa_get_next_attr_value()

CONFORMING TO
Distributed Resource Management Application API Specification 1.0
(GWD-R)

SECTION 6. job submission

```
NAME
   drmaa_run_job, drmaa_run_bulk_jobs - submit single and bulk jobs

SYNOPSIS

   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);

   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);

DESCRIPTION
   These two functions are used for job submission to a DRM system.

   The drmaa_run_job() submits a single job with the attributes
         defined in the job template 'jt'. On success up to
         'job_id_len' bytes of the job identifier are returned into
         the buffer 'job_id'.

   The drmaa_run_bulk_jobs() submits a set of parametric jobs that
         can be run concurrently. For each parametric job the
         attributes defined in the job template 'jt' are used. Each
         job of the set is identical except of it's index. The first
         parametric job gets 'start' as index, the next one gets
         'start' + 'incr' etc. until 'end'. The smallest 'start' is
         1, the largest 'end' is 2^31-1. The 'start' value must be
         lower or equal than 'end' and only positive index numbers
         are accepted. The index number can be determined by the job
         in an implementation specific fashion.
On success
   a job id string vector containing job identifiers is returned
   into 'jobids'. The job identifiers in the job id string vector
   can be extracted using drmaa_get_next_job_id(). The  caller is
   responsible for releasing the job id string vector returned into
   'jobids' using drmaa_release_job_ids().

RETURNS
   Upon successful completion drmaa_run_job() and
   drmaa_run_bulk_jobs()
   return DRMAA_ERRNO_SUCCESS, otherwise corresponding DRMAA error
   codes are returned.

ERRORS
   If any error condition occurs, drmaa_run_job() and
   drmaa_run_bulk_jobs()  shall provide up to 'error_diag_len'
   characters of error related diagnosis information in the buffer
   'error_diagnosis' and return one of following error codes:

   DRMAA_ERRNO_TRY_LATER
      Could not pass job now to DRM system. A retry may succeed
         however (saturation).

   DRMAA_ERRNO_DENIED_BY_DRM
      The DRM system rejected the job. The job will never be
```

```
            accepted due to DRM configuration or job template settings.

      DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE
         Could not contact DRM system for this request.

      DRMAA_ERRNO_AUTH_FAILURE
         The specified request is not processed successfully due to
            authorization failure.

      DRMAA_ERRNO_NO_MEMORY
         The system is unable to allocate resources.

      DRMAA_ERRNO_INTERNAL_ERROR
         Unexpected or internal error.

      DRMAA_ERRNO_INVALID_ARGUMENT
         The input value for an argument is invalid.

      DRMAA_ERRNO_NO_ACTIVE_SESSION
         Routine failed because there is no active session.

   SEE ALSO
      drmaa_get_next_job_id(), drmaa_release_job_ids()

   EXAMPLE

   CONFORMING TO
      Distributed Resource Management Application API Specification 1.0
            (GWD-R)

SECTION 7. job status and control

   NAME

      drmaa_control, drmaa_job_ps - control a job and obtain a job's
            status

   SYNOPSIS
      #include "drmaa.h"

      int drmaa_control(const char *jobid, int action, char
            *error_diagnosis, size_t error_diag_len);

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

   DESCRIPTION
      The drmaa_control() function allows the job specified by jobid to
      be controlled according to action, whose 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() call returns after the DRM system has acknowledged the command, not necessarily after the desired action has been performed. f jobid is DRMAA_JOB_IDS_SESSION_ALL, then this function performs action on all jobs submitted during this session at the moment it is called.

The drmaa_job_ps() function fills remote_ps with the program status of the job identified by job_id. The possible values of a program's staus 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 DRMAA_PS_FAILED status.

RETURNS

All functions return DRMAA_ERRNO_SUCCESS on success.

ERRORS

If any error condition occurs, drmaa_control() and drmaa_job_ps() shall provide up to error_diag_len characters of error related diagnosis information in the buffer error_diagnosis and return an appropriate error code from those described below.

The drmaa_control() function may return DRMAA_ERRNO_AUTH_FAILURE, DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE, DRMAA_ERRNO_HOLD_INCONSISTENT_STATE, DRMAA_ERRNO_INTERNAL_ERROR, DRMAA_ERRNO_INVALID_JOB, DRMAA_ERRNO_NO_MEMORY, DRMAA_ERRNO_RELEASE_INCONSISTENT_STATE, DRMAA_ERRNO_RESUME_INCONSISTENT_STATE or DRMAA_ERRNO_SUSPEND_INCONSISTENT_STATE.

The drmaa_job_ps() function may return DRMAA_ERRNO_AUTH_FAILURE, DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE, DRMAA_ERRNO_INTERNAL_ERROR, DRMAA_ERRNO_INVALID_JOB or DRMAA_ERRNO_NO_MEMORY.

DRMAA_ERRNO_AUTH_FAILURE

The specified request is not processed successfully due to authorization failure.

DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE

Could not contact DRM system for this request.

DRMAA_ERRNO_HOLD_INCONSISTENT_STATE

The job cannot be moved to a HOLD state.

DRMAA_ERRNO_INTERNAL_ERROR

```
         Unexpected or internal error.

      DRMAA_ERRNO_INVALID_JOB
         A jobid is invalid.

      DRMAA_ERRNO_NO_MEMORY
         The system is unable to allocate the required memory.

      DRMAA_ERRNO_RELEASE_INCONSISTENT_STATE
         The job is not in a HOLD state.

      DRMAA_ERRNO_RESUME_INCONSISTENT_STATE
         The job has not been suspended.

      DRMAA_ERRNO_SUSPEND_INCONSISTENT_STATE
         The job has not been running and cannot be suspended.

   SEE ALSO
      drmaa_run_job()

   CONFORMING TO
      Distributed Resource Management Application API Specification 1.0
(GWD-R)


SECTION 8. synchronize, wait

   NAME
      drmaa_synchronize, drmaa_wait, drmaa_wifexited, drmaa_wifsignaled,
      drmaa_wtermsig, drmaa_wcoredump, drmaa_wifaborted - synchronize
            and wait operations

   SYNOPSIS
      #include "drmaa.h"

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

      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);

      int drmaa_wifexited(int *exited, int stat, char *error_diagnosis,
                  size_t error_diag_len);

      int drmaa_wexitstatus(int *exit_status, int stat, char
                  *error_diagnosis, size_t error_diag_len);

      int drmaa_wifsignaled(int *signaled, int stat, char
                  *error_diagnosis, size_t error_diag_len);

      int drmaa_wtermsig(char *signal, size_t signal_len, int stat,
                  char *error_diagnosis, size_t error_diag_len);

      int drmaa_wcoredump(int *core_dumped, int stat, char
```

```
                *error_diagnosis, size_t error_diag_len);

    int drmaa_wifaborted(int *aborted, int stat, char
                *error_diagnosis, size_t error_diag_len);
```

DESCRIPTION

The drmaa_synchronize() function blocks until all jobs specified by job_ids have completed execution. Use DRMAA_JOB_IDS_SESSION_ALL as job_ids in order to wait for all jobs submitted before this function is called. The function waits for the number of seconds specified by timeout, where a timeout of DRMAA_TIMEOUT_WAIT_FOREVER is an infinite amount of time and DRMAA_TIMEOUT_NO_WAIT returns immediately. If this function returns before timeout seconds either all the jobs have been waited on or there was an interrupt. This function returns DRMAA_ERRNO_EXIT_TIMEOUT if it exits due to timeout before all jobs have completed. The dispose parameter specifies how to treat the rusage data of the waited jobs; if false, the information remains available via drmaa_wait(), otherwise it is discarded.

The drmaa_wait() function waits for the job specified by job_id to either finish executing or fail. If job_id is DRMAA_JOB_IDS_SESSION_ANY, drmaa_wait() waits for any job submitted this session. Like drmaa_synchronize(), drmaa_wait() waits for the number of seconds specified by timeout, where timeout may be DRMAA_TIMEOUT_WAIT_FOREVER or DRMAA_TIMEOUT_NO_WAIT.

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 encompasses information about the condition under which the job terminated, and rusage with an array of <name>=<value> strings that describe the amount of resources consumed by the job. The stat parameter is further described below. The rusage parameter's values may be accessed via drmaa_get_next_attr_value().

If drmaa_wait() exits before timeout, either the job has been successfully waited or there was an interrupt. If successfully waited, the jobs rusage information has been reaped, and further calls to drmaa_wait() with this job_id will return DRMAA_ERRNO_INVALID_JOB. If drmaa_wait() exists due to timeout, DRMAA_ERRNO_EXIT_TIMEOUT is returned and no rusage information is reaped.

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, job_id_out, through the following functions: drmaa_wifexited(), drmaa_wexitstatus(), drmaa_wifsignaled(), drmaa_wtermsig(), drmaa_wcoredump() and drmaa_wifaborted().

The drmaa_wifexited() function evaluates into exited a non-zero value if stat was returned for a job that termined normally. In this case, more information can be provided about the job by the drmaa_wifsignaled() and drmaa_wcoredump() functions. Exited is

filled with zero if either the job's termination state is unknown or no exit status of a normally terminated job is available.

The drmaa_wexitstatus() function evaluates into exit_status the exit code of the job provided that drmaa_wifexited()'s exited is non-zero.

The drmaa_wifsignaled() function evaluates into signaled a non-zero value if stat was returned for a job that terminated due to the receipt of a signal. A zero value indicates either the job did not terminate due to a signal or it is not known if the job terminated due to a signal.

The drmaa_wtermsig() function fills signal with up to signal_len characters of the signal name that caused the termination of the job provided that the call to drmaa_wifsignaled() with this stat returned non-zero. For signals declared by POSIX, the symbolic names are returned.

The drmaa_wcoredump() function fills core_dumped with a non-zero value provided that drmaa_wifsignaled()'s signaled is non-zero and a core image of the terminated job was created.

The drmaa_wifaborted() function fills aborted with a non-zero value if stat was returned for a job that ended before entering the running state.

RETURNS

All functions return DRMAA_ERRNO_SUCCESS on success.

ERRORS

If any error condition occurs, all functions shall provide up to error_diag_len characters of error related diagnosis information in the buffer error_diagnosis and return an appropriate error code from those described below.

The drmaa_synchronize() function may return DRMAA_ERRNO_NO_MEMORY, DRMAA_ERRNO_INTERNAL_ERROR, DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE, DRMAA_ERRNO_AUTH_FAILURE or DRMAA_ERRNO_INVALID_JOB.

The drmaa_wait() function may return DRMAA_ERRNO_NO_MEMORY, DRMAA_ERRNO_INTERNAL_ERROR, DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE, DRMAA_ERRNO_AUTH_FAILURE, DRMAA_ERRNO_NO_RUSAGE or DRMAA_ERRNO_INVALID_JOB.

DRMAA_ERRNO_AUTH_FAILURE
: The specified request is not processed successfully due to authorization failure.

DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE
: Could not contact DRM system for this request.

DRMAA_ERRNO_INTERNAL_ERROR
: Unexpected or internal error.

DRMAA_ERRNO_INVALID_JOB
: A jobid is invalid.

```
      DRMAA_ERRNO_NO_MEMORY
         The system is unable to allocate the required memory.

      DRMAA_ERRNO_NO_RUSAGE
         The job has finished but no rusage and stat data is available.

   SEE ALSO drmaa_get_next_attr_value(), drmaa_release_attr_values()

   CONFORMING TO
      Distributed Resource Management Application API Specification 1.0
(GWD-R)


SECTION 9. auxilliary

   NAME
      drmaa_strerror, drmaa_get_contact, drmaa_version,
      drmaa_get_DRM_system, drmaa_get_DRMAA_implementaion - auxillary
                              operations

   SYNOPSIS
      #include "drmaa.h"

      const char *drmaa_strerror(int drmaa_errno);

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

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

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

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

   DESCRIPTION
      The drmaa_strerror() function returns the error string describing
      the DRMAA error number drmaa_errno.

      The drmaa_get_contacts() function, if called before drmaa_init(),
      returns the list of default DRMAA implementation contacts strings,
      one per each DRM implementation provided.  These contact strings
      are delimited by commas.  If called after drmaa_init(),
      drmaa_get_contacts() returns the contact string of the DRM system
      for which the library has been initialized.

      The drmaa_version() function sets major and minor to the major
      and minor versions of the DRMAA library.

      The drmaa_get_DRM_system() function, if called before
      drmaa_init(), returns a comma delimited DRM systems string, one
      per each DRM system implementation provided.  If called after
      drmaa_init(), it returns the selected DRM system.
```

```
      The drmaa_get_DRMAA_implementation() function, if called before
      drmaa_init(), returns a command delimited DRMAA implementations
      string, one per each DRM system implementation provided.  If
      called after drmaa_init(), it returns the selected DRMAA
      implementation.

   RETURNS
      The drmaa_strerror() function returns the appropriate string given a
valid
      drmaa_errno, otherwise it returns NULL.

      The drmaa_get_contact(), drmaa_version(), drmaa_get_DRM_system(),
      and drmaa_get_DRMAA_implmentation() functions always return
      DRMAA_ERRNO_SUCCESS.

   SEE ALSO
      drmaa_init()

   CONFORMING TO
      Distributed Resource Management Application API Specification 1.0
(GWD-R)
```

3. C binding example

The C test program below serves as an example of an application that uses the DRMAA C binding interface. It illustrates submission of both single and bulk remote jobs. After submission drmaa_synchronize() call is used to synchronize the remote jobs execution. The call returns after all the jobs have finished executing. Finally, drmaa_wait() call is used to retrieve and print out the remote jobs execution information.

A full path for the remote command is passed as the first argument to the test program. That value is directly used as “drmaa_remote_command” job template attribute. The C binding example uses value “5” as a first argument to the job template vector attribute “drmaa_v_argv”. Passing “/bin/sleep” as a first argument to the test program will for example cause 32 sleep jobs to be run that sleep for 5 seconds each before finishing execution. Note that we expect to find “/bin/sleep” command on all of the remote nodes.

```
#include <stdio.h>;
#include <unistd.h>;
#include <string.h>;
#include "drmaa.h"
#define JOB_CHUNK 8
#define NBULKS 3
static drmaa_job_template_t *create_job_template(const char *job_path,

int seconds, int as_bulk_job);
int main(int argc, char *argv[])
{
      char diagnosis[DRMAA_ERROR_STRING_BUFFER];
      const char *all_jobids[NBULKS*JOB_CHUNK + JOB_CHUNK+1];
      char jobid[100];
      int drmaa_errno, i, pos = 0;
      const char *job_path;
      drmaa_job_template_t *jt;
      drmaa-wg@gridforum.org 8
            if (argc < 2) {
                  fprintf(stderr, "usage: example path-to-job\n");
```

```
      return 1;
}
job_path = argv[1];
if (drmaa_init(NULL, diagnosis, sizeof(diagnosis)-1) !=
      DRMAA_ERRNO_SUCCESS) {
      fprintf(stderr, "drmaa_init() failed: %s\n", diagnosis);
      return 1;
}
/* submit some bulk jobs */
if (!(jt = create_job_template(job_path, 5, 1))) {
      fprintf(stderr, "create_job_template() failed\n");
      return 1;
}
for (i=0; I < NBULKS; i++) {
      drmaa_job_ids_t *jobids;
      int j;
      while ((drmaa_errno=drmaa_run_bulk_jobs(&jobids, jt, 1,
            JOB_CHUNK, 1, diagnosis, sizeof(diagnosis)- 1))
            ==DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE) {
            fprintf(stderr, "drmaa_run_bulk_jobs() failed –
                  retry: %s\n", diagnosis);
            sleep(1);
      }
      if (drmaa_errno != DRMAA_ERRNO_SUCCESS) {
            fprintf(stderr, "drmaa_run_bulk_jobs()
                  failed: %s\n", diagnosis);
            return 1;
      }
      printf("submitted bulk job with jobids:\n");
      for (j=0; j < JOB_CHUNK; j++) {
            drmaa_get_next_job_id(jobids, jobid,
                  sizeof(jobid)-1);
            all_jobids[pos++] = strdup(jobid);
            printf("\t \"%s\"\n", jobid);
      }
      drmaa_release_job_ids(jobids);
}
drmaa_delete_job_template(jt, NULL, 0);
/* submit some sequential jobs */
if (!(jt = create_job_template(job_path, 5, 0))) {
      fprintf(stderr, "create_sleeper_job_template()
                        failed\n");
      return 1;
}
for (i=0; I JOB_CHUNK; i++) {
      while ((drmaa_errno=drmaa_run_job(jobid,
            sizeof(jobid)-1, jt,
            diagnosis, sizeof(diagnosis)-1)) ==
            DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE) {
            fprintf(stderr, "drmaa_run_job() failed –
                  retry: %s\n", diagnosis);
            sleep(1);
      }
      if (drmaa_errno != DRMAA_ERRNO_SUCCESS) {
            fprintf(stderr, "drmaa_run_job() failed: %s\n",
                  diagnosis);
            return 1;
            drmaa-wg@gridforum.org 9
      }
      printf("\t \"%s\"\n", jobid);
      all_jobids[pos++] = strdup(jobid);
}
/* set string array end mark */
all_jobids[pos] = NULL;
```

```
        drmaa_delete_job_template(jt, NULL, 0);
        /* synchronize with all jobs */
        drmaa_errno = drmaa_synchronize(all_jobids,
              DRMAA_TIMEOUT_WAIT_FOREVER, 0,
              diagnosis, sizeof(diagnosis)-1);
        if (drmaa_errno != DRMAA_ERRNO_SUCCESS) {
              fprintf(stderr,
                    "drmaa_synchronize(DRMAA_JOB_IDS_SESSION_ALL,
                    dispose) failed: %s\n", diagnosis);
                    return 1;
        }
        printf("synchronized with all jobs\n");
        /* wait all those jobs */
        for (pos=0; pos < NBULKS*JOB_CHUNK + JOB_CHUNK; pos++) {
              int stat;
              int aborted, exited, exit_status, signaled;
              drmaa_errno = drmaa_wait(all_jobids[pos], jobid,
                    sizeof(jobid)-1, &stat,
                    DRMAA_TIMEOUT_WAIT_FOREVER, NULL,
                    diagnosis, sizeof(diagnosis)-1);
              if (drmaa_errno != DRMAA_ERRNO_SUCCESS) {
                    fprintf(stderr, "drmaa_wait(%s) failed: %s\n",
                          all_jobids[pos], diagnosis);
                    return 1;
              }
              /* report how job finished */
              drmaa_wifaborted(&aborted, stat, NULL, 0);
              if (aborted)
                    printf("job \"%s\" never ran\n", all_jobids[pos]);
              else {
                    drmaa_wifexited(&exited, stat, NULL, 0);
                    if (exited) {
                          drmaa_wexitstatus(&exit_status, stat, NULL,
                                0);
                          printf("job \"%s\" finished regularly with
                                exit status d\n",
                                all_jobids[pos], exit_status);
                    } else {
                          drmaa_wifsignaled(&signaled, stat, NULL,
                                0);
                          if (signaled) {
                                char termsig[DRMAA_SIGNAL_BUFFER+1];
                                drmaa_wtermsig(termsig,
                                DRMAA_SIGNAL_BUFFER, stat,
                                NULL, 0);
                                printf("job \"%s\" finished due to
                                signal %s\n",
                                      all_jobids[pos], termsig);
                          } else
                                printf("job \"%s\" finished with
                                            unclear conditions\n",
                                all_jobids[pos]);
                    }
              }
        }

        if (drmaa_exit(diagnosis, sizeof(diagnosis)-1) !=
              DRMAA_ERRNO_SUCCESS) {
              fprintf(stderr, "drmaa_exit() failed: %s\n", diagnosis);
              return 1;
        }
        return 0;
}
static drmaa_job_template_t *create_job_template(const char *job_path,
```

```
int seconds, int as_bulk_job)
{
      const char *job_argv[2];
      drmaa_job_template_t *jt = NULL;
      char buffer[100];
      if (drmaa_allocate_job_template(&jt, NULL, 0)!=DRMAA_ERRNO_SUCCESS)
            return NULL;
      /* run in users home directory */
      drmaa_set_attribute(jt, DRMAA_WD, DRMAA_PLACEHOLDER_HD, NULL, 0);
      /* the job to be run */
      drmaa_set_attribute(jt, DRMAA_REMOTE_COMMAND, job_path, NULL, 0);
      /* the job's arguments */
      sprintf(buffer, "%d", seconds);
      job_argv[0] = buffer;
      job_argv[1] = NULL;
      drmaa_set_vector_attribute(jt, DRMAA_V_ARGV, job_argv, NULL, 0);
      /* join output/error file */
      drmaa_set_attribute(jt, DRMAA_JOIN_FILES, "y", NULL, 0);
      /* path for output */
      if (!as_bulk_job)
            drmaa_set_attribute(jt, DRMAA_OUTPUT_PATH,
            DRMAA_PLACEHOLDER_HD"/DRMAA_JOB", NULL, 0);
      else
            drmaa_set_attribute(jt, DRMAA_OUTPUT_PATH,
            DRMAA_PLACEHOLDER_HD"/DRMAA_JOB."DRMAA_PLACEHOLDER_INCR,
            NULL, 0);
      return jt;
}
```

4. Security Considerations

The DRMAA API does not specifically assume the existence of GRID Security infrastructure. The scheduling scenario described herein presumes that security is handled at the point of job authorization/execution on a particular resource. It is assumed that credentials owned by the process using the API are used by the DRMAA implementation to prevent abuse of the interface. In order to not unnecessarily restrict the spectrum of usable credentials, no explicit interface is defined for passing credentials.

It is conceivable an authorized but malicious user could use a DRMAA implementation or a DRMAA enabled application to saturate a DRM system with a flood of requests. Unfortunately for the DRM system this case is not distinguishable from the case of an authorized good-natured user that has many jobs to be processed. For this case DRMAA defines the DRMAA_ERRNO_TRY_LATER return code to allow a DRM system to reject requests and properly indicate DRM saturation.

DRMAA implementers should guard against buffer overflows that could be exploited through DRMAA enabled interactive applications or web portals. Implementations of the DRMAA API will most likely require a network to coordinate subordinate DRMS, however the API makes no assumptions about the security posture provided the networking environment. Therefore, application developers should further consider the security implications of "on-the-wire" communications.

For environments that allow remote or protocol based DRMAA clients DRMAA should consider implementing support for secure transport layers to prevent man in the middle attacks. DRMAA does not impose any security requirements on its clients.

Author Information

Roger Brobst
rbrobst@cadence.com
Cadence Design Systems, Inc
555 River Oaks Parkway
San Jose, CA 95134

Nicholas Geib
njgeib@wisc.edu
University of Wisconsin Madison
USA

Andreas Haas
andreas.haas@sun.com
Sun Microsystems GmbH
Dr.-Leo-Ritter-Str. 7
D-93049 Regensburg
Germany

Hrabri L. Rajic
hrabri.rajic@intel.com
Intel Americas Inc.
1906 Fox Drive
Champaign, IL 61820

John Tollefsrud
j.t@sun.com
Sun Microsystems
18 Network Circle, UMPK18-211
Menlo Park, CA 94025

Intellectual Property Statement

or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the GGF Secretariat.

The GGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the GGF Executive Director.

Full Copyright Notice