GFD-E

Daniel Templeton, Sun Microsystems (maintainer)

Distributed Resource Management
Application API (DRMAA) Working Group

February, 2005

Distributed Resource Management Application API C Bindings Experience Document

Status of This Memo

This memo is a Global Grid Forum Experimental Document (GFD-E) 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 experience gained from implementation work on the Distributed Resource Management Application API (DRMAA) C binding. The document is based on the implementation work of the DRMAA GWD-R document.

Table of Contents

1. Overview

This document provides comments on the Distributed Resource Management Application API C Bindings document derived from implementation and user experience.

2. Conventions

In this document, the following conventions are used:

- `C Language Elements` are represented in a fixed-width font.
- *References to Parameters* to functions are represented in itallics.
- `Parameter and Error Code Names` in parameter and error code definitions are represented in a fixed-width font.
- *Attribute Names* for job templates are represented in itallics.

2.1 API Conventions

The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” are to be interpreted as described in RFC-2119 [RFC 2119].

In function definitions, the word “return” is often used to denote that a value is received from the function. This word does not necessarily mean that the returned value is received via the function's return value. The information shown in the function signature listings always takes precedence and should be used as a basis for interpreting the textual function definitions.

2.2 Abbreviations

The following abbreviations are used in this document:

API	Application Programming Interface
DRM	Distributed Resource Manager
DRMS	Distributed Resource Management System
DRMAA	Distributed Resource Management Application API
ISV	Independent Software Vendor

3. Experience

The information contained in this document stems from the experience gained while implementing the Distributed Resource Management Application API C Bindings 0.95 specification for the Sun Grid Engine 6.0 DRMS.

3.1 Ease of Implementation

The Distributed Resource Management Application API Specification 1.0 is written with a particular focus on implementability for DRM vendors. This focus makes implementing the Distributed Resource Management Application API C Bindings 0.95 specification an easier task than it may otherwise have been. In particular, the life cycle model of job exit information as defined by the drmaa_wait() and drmaa_synchronize() routines greatly simplifies the implementation of the specification.

The concept of the job template is very well constructed to provide maximum flexibility for the specification implementers. The job template can be easily translated into an SGE internal job structure.

The multiple return values from the drmaa_get_DRM_system(), drmaa_get_contact() and drmaa_get_DRMAA_implementation(), while cumbersome on paper, are easily implemented in a single-DRM DRMAA implementation: no matter what the context, each method always returns the same string.

The multi-threaded aspect of the Distributed Resource Management Application API Specification 1.0 makes building an implementation a little more difficult. Accurate data access synchronization must be used to ensure different threads do not interfere with each other. The drmaa_init() and drmaa_exit() routines proved to be the most challenging in this regard, as they must affect the behavior of all threads accessing the DRMAA implementation.

3.2 Ease of Use

Developers using the Sun Grid Engine implementation of the Distributed Resource Management Application API C Bindings 0.95 specification are faced with a simple API which is complicated to use.

The API itself is quite simple. The majority of the functionality offered by DRMAA is encapsulated in four functions and one data structure. Learning the API is not complicated and can be done in a short period of time. The concepts in the API are easy to understand, and pose no great challenge for experienced developers.

The use of the API, however, can be less than straight-forward. Several semantic issues make the API a little thorny and make the resulting DRMAA-enabled application's source code rather ugly.

The first issue is error handling. Because every function returns an error code as its return value, any actual return values have to be returned in pass-by-reference parameters. This complicates the signature for the DRMAA routines, with drmaa_wait() being a prime example. It also complicates applications which use these functions. For example, because the return value from the drmaa_wifexited(), drmaa_wifsignaled(), and drmaa_ifaborted() is an error code and not the boolean value, these functions cannot be chained in an if structure. Take the following example:

```
int exited;

drmaa_ifexited (&exited, &stat, ...);

if (exited) {
   int signaled;

   drmaa_ifsignaled (&signaled, &stat, ...);

   if (signaled) {
      int aborted;

      drmaa_ifaborted (&aborted, &stat, ...);

      if (aborted) {
         ...
```

```
        } else {
           ...
        }
     } else {
        ...
     }
} else {
   ...
}
```

would be much clearer if these functions returned a boolean value. The result might be something like the following:

```
if (drmaa_ifexited (&stat, ...)) {
   ...
} else if (drmaa_ifsignaled (&stat, ...)) {
   ...
} else if (drmaa_ifaborted (&stat, ...)) {
   ...
} else {
   ...
}
```

which is both easier to understand and debug.

Another issue with the API is that each routine assumes the caller will allocate the required space for any string buffers. This decision has two negative impacts. First, it forces the developer to pick a number of the size of the buffer, which in almost every case results in either allocating too much space, or less often, in not allocating enough space, resulting in a truncated result or an error. Second, it requires every routine which accepts a string buffer to also take a parameter which states the size of the buffer. This extra parameter complicates the routine signatures. In the case of drmaa_wait(), this issue adds **three** parameters to the routine's signature. A less obtrusive way to dealing with string buffers would be for the DRMAA routines to allocate the space for the buffer, with the assumption that the DRMAA-enabled calling application will free that buffer space when the string is no longer needed. This alternative solution can, however, cause problems in systems where memory is allocated in such a way that the free(3) system call cannot release it properly.

3.3 Missing Functionality

An important piece which is missing from the Distributed Resource Management Application API Specification 1.0 is a way for the drmaa_wait() routine to provide a reason why a job aborted. Without such functionality, DRMAA offers the developer no way to diagnose problems in running jobs. This limitation has a serious impact on the usability of a DRMAA-enabled application.

A second important piece which is missing from the Distributed Resource Management Application API Specification 1.0 is the notion of an error state. In the Sun Grid Engine 6.0 implementation, it is possible for a job to be placed into an error state. A job in an error state has not failed. It is waiting for an administrator to resolve an issue which is preventing it from running. Because the Distributed Resource Management Application API Specification 1.0 did not forsee an error state, such jobs appear to a DRMAA-enabled application as being in the running state.

While this assessment is not incorrect, it robs the DRMAA-enabled application of the ability to notify the end user of the problem or to deal with the problem itself.

A third issue is modular pluggability. The Distributed Resource Management Application API C Bindings 0.95 specification does not clearly specify how the DRMAA implementation library should be linked or named, resulting in a complete lack of modular pluggability. This issue is fixed in the 0.98 version of the specification.

A fourth issue is that Distributed Resource Management Application API Specification 1.0 does not provide for a way for a DRMAA-enabled application to receive notification of job state changes other than the move from running to done or failed. DRMAA-enabled applications need the ability to receive job state change notifications, particularly the move from pending to running, without having to implement polling using drmaa_job_ps(). Currently, applications are forced to rely on native DRM functionality.

Another missing piece is the DRMAA_PS_USER_SYSTEM_SUSPENDED state. This state was erroneously left out of the Distributed Resource Management Application API Specification 1.0, preventing binding specifications and hence implementations from offering that state as a result from the drmaa_job_ps() routine.

4. Application Programming Interface Proposal

What follows is a sample specification of the Distributed Resource Management Application API C Bindings as might represent a logical next step. This specification is based on version 0.98 of the Distributed Resource Management Application API C Bindings document.

4.1 Compile Time Symbols

4.1.1 Opaque Data Types

Five opaque data types SHALL be defined by a C binding implementation. They are:

```
typedef struct drmaa_job_template_s drmaa_job_template_t;
typedef struct drmaa_job_info_s drmaa_job_info_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;
```

The `drmaa_job_template_s` data type holds the attribute values for a job template. A job template is used to define the characteristics of a job to be submitted via the drmaa_run_job() or drmaa_run_bulk_jobs() functions. See sections 4.5.1 and 4.5.2 for more details.

The `drmaa_job_info_s` data type holds the exit information for a completed job. The exit information structure is returned from the drmaa_wait() function. See section 4.6.4 for more details.

The remaining three data types are used to store and iterate through name, value, and job id data. Special functions are defined for extracting values from the structures and releasing the structures when they are no longer needed. See section **Error! Reference source not found.** for more details.

In order to preclude the unintended access to these structures' internal data members, the DRMAA header file SHALL NOT include the `struct` definitions. By leaving the `struct`

definitions out of the header file, it ensures that the structures' internal data members can only be accessed via the access functions described in section **Error! Reference source not found.**.

4.1.2 C Preprocessor Directives for String Handling

A DRMAA header file SHALL define the following preprocessor directives:

```
#define DRMAA_ATTR_BUFFER            1024
#define DRMAA_CONTACT_BUFFER         1024
#define DRMAA_DRM_SYSTEM_BUFFER      1024
#define DRMAA_DRMAA_IMPL_BUFFER      1024
#define DRMAA_ERROR_STRING_BUFFER    1024
#define DRMAA_JOBNAME_BUFFER         1024
#define DRMAA_SIGNAL_BUFFER            32
```

These preprocessor directives define default lengths which may be used by developers in the creation of `char*` variables. The sizes defined by these directives are the recommended **minimum** lengths for buffer variables used to store the corresponding data values. If buffer variables of shorter length are used, the DRMAA implementation MUST either truncate the data string to be stored in the buffer variable or return the DRMAA_ERRNO_INVALID_ARGUMENT error.

4.1.2.1 DRMAA_ATTR_BUFFER

The DRMAA_ATTR_BUFFER directive is the default length for the attribute names used by drmaa_set_attribute(), drmaa_set_vector_attribute(), drmaa_get_attribute(), drmaa_get_vector_attribute(), and drmaa_get_attribute_names().

4.1.2.2 DRMAA_CONTACT_BUFFER

The DRMAA_CONTACT_BUFFER directive is the default length for the contact string used by drmaa_init().

4.1.2.3 DRMAA_DRM_SYSTEM_BUFFER

The DRMAA_DRM_SYSTEM_BUFFER directive is the default length for DRM system name returned by drmaa_get_DRM_system().

4.1.2.4 DRMAA_DRMAA_IMPL_BUFFER

The DRMAA_DRMAA_IMPL_BUFFER directive is the default length for the DRMAA implementation name returned by drmaa_get_DRMAA_implementation().

4.1.2.5 DRMAA_ERROR_STRING_BUFFER

The DRMAA_ERROR_STRING_BUFFER directive is the default length for error strings returned by most DRMAA functions.

4.1.2.6 DRMAA_JOBNAME_BUFFER

The DRMAA_JOBNAME_BUFFER directive is the default length for the job identifiers used by drmaa_run_job(), drmaa_run_bulk_jobs(), drmaa_wait(), drmaa_synchronize(), drmaa_control(), and drmaa_job_ps().

4.1.2.7 DRMAA_SIGNAL_BUFFER

The DRMAA_SIGNAL_BUFFER directive is the default length for the signal name returned by drmaa_wtermsig().

4.1.3 C Preprocessor Directives for Control Operations

A DRMAA header file SHALL define the following preprocessor directives:

```
#define DRMAA_TIMEOUT_NO_WAIT           0
#define DRMAA_TIMEOUT_WAIT_FOREVER      -1
#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
#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
#define DRMAA_JOB_IDS_SESSION_ALL       "DRMAA_JOB_IDS_SESSION_ALL"
#define DRMAA_JOB_IDS_SESSION_ANY       "DRMAA_JOB_IDS_SESSION_ANY"
```

4.1.3.1 DRMAA_TIMEOUT_NO_WAIT

The DRMAA_TIMEOUT_NO_WAIT directive is used as to indicate to drmaa_wait() and drmaa_synchronize() that the implementation should not block if the requested job exit status information is not available.

4.1.3.2 DRMAA_TIMEOUT_WAIT_FOREVER

The DRMAA_TIMEOUT_WAIT_FOREVER directive is used as to indicate to drmaa_wait() and drmaa_synchronize() that the implementation should block indefinitely until the requested job exit status information is available.

4.1.3.3 DRMAA_PS_UNDETERMINED

The DRMAA_PS_UNDETERMINED directive is used by drmaa_job_ps() to indicate that status of the requested job cannot be determined.

4.1.3.4 DRMAA_PS_QUEUED_ACTIVE

The DRMAA_PS_QUEUED_ACTIVE directive is used by drmaa_job_ps() to indicate that the requested job's status is queued and active.

4.1.3.5 DRMAA_PS_SYSTEM_ON_HOLD

The DRMAA_PS_SYSTEM_ON_HOLD directive is used by drmaa_job_ps() to indicate that the requested job has been placed in a hold state by the system or administrator.

4.1.3.6 DRMAA_PS_USER_ON_HOLD

The DRMAA_PS_USER_ON_HOLD directive is used by drmaa_job_ps() to indicate that the requested job has been placed in a hold state by the user.

4.1.3.7 DRMAA_PS_USER_SYSTEM_ON_HOLD

The DRMAA_PS_USER_SYSTEM_ON_HOLD directive is used by drmaa_job_ps() to indicate that the requested job has been placed in a hold state by the system or administrator and the user.

4.1.3.8 DRMAA_PS_RUNNING

The DRMAA_PS_RUNNING directive is used by drmaa_job_ps() to indicate that the requested job is running

4.1.3.9 DRMAA_PS_SYSTEM_SUSPENDED

The DRMAA_PS_SYSTEM_SUSPENDED directive is used by drmaa_job_ps() to indicate that the requested job has been placed in a suspend state by the system or administrator.

4.1.3.10 DRMAA_PS_USER_SUSPENDED

The DRMAA_PS_USER_SUSPENDED directive is used by drmaa_job_ps() to indicate that the requested job has been placed in a suspend state by the user.

4.1.3.11 DRMAA_PS_USER_SYSTEM_SUSPENDED

The DRMAA_PS_USER_SYSTEM_SUSPENDED directive is used by drmaa_job_ps() to indicate that the requested job has been placed in a suspend state by the system or administrator and user.

4.1.3.12 DRMAA_DONE

The DRMAA_PS_DONE directive is used by drmaa_job_ps() to indicate that the requested job has successfully completed.

4.1.3.13 DRMAA_FAILED

The DRMAA_PS_FAILED directive is used by drmaa_job_ps() to indicate that the requested job has terminated execution abnormally.

4.1.3.14 DRMAA_CONTROL_SUSPEND

The DRMAA_CONTROL_SUSPEND directive is used to indicate to drmaa_control() that the requested job should be placed in a user suspend state.

4.1.3.15 DRMAA_CONTROL_RESUME

The DRMAA_CONTROL_RESUME directive is used to indicate to drmaa_control() that the requested job should be resumed from a user suspend state.

4.1.3.16 DRMAA_CONTROL_HOLD

The DRMAA_CONTROL_HOLD directive is used to indicate to drmaa_control() that the requested job should be placed into a user hold state.

4.1.3.17 DRMAA_CONTROL_RELEASE

The DRMAA_CONTROL_RELEASE directive is used to indicate to drmaa_control() that the requested job should the released from a user hold state.

4.1.3.18 DRMAA_CONTROL_TERMINATE

The DRMAA_CONTROL_TERMINATE directive is used to indicate to drmaa_control() that the requested job should be terminated.

4.1.3.19 DRMAA_JOB_IDS_SESSION_ALL

The DRMAA_JOB_IDS_SESSION_ALL directive is used to indicate to drmaa_control() and drmaa_synchronize() that all jobs currently active in the session should be the operation's target.

4.1.3.20 DRMAA_JOB_IDS_SESSION_ANY

The DRMAA_JOB_IDS_SESSION_ANY directive is used to indicate to drmaa_wait() that any job currently active in the session should be the operation's target.

4.1.4 C Preprocessor Directives for Job Template Compilation

A DRMAA header file SHALL define the following preprocessor directives:

```
#define DRMAA_BLOCK_EMAIL              "drmaa_block_email"
#define DRMAA_DEADLINE_TIME            "drmaa_deadline_time"
#define DRMAA_DURATION_HLIMIT          "drmaa_duration_hlimit"
#define DRMAA_DURATION_SLIMIT          "drmaa_duration_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"
#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 more information about job template attributes, see sections **Error! Reference source not found.** and **Error! Reference source not found.**.

4.1.4.1 DRMAA_BLOCK_EMAIL

The DRMAA_BLOCK_EMAIL directive is used to represent the *drmaa_block_email* attribute.

4.1.4.2 DRMAA_DEADLINE_TIME

The DRMAA_DEADLINE_TIME directive is used to represent the *drmaa_deadline_time* attribute.

4.1.4.3 DRMAA_DURATION_HLIMIT

The DRMAA_DURATION_HLIMIT directive is used to represent the *drmaa_duration_hlimit* attribute.

4.1.4.4 DRMAA_DURATION_SLIMIT

The DRMAA_DURATION_SLIMIT directive is used to represent the *drmaa_duration_slimit* attribute.

4.1.4.5 DRMAA_ERROR_PATH

The DRMAA_ERROR_PATH directive is used to represent the *drmaa_error_path* attribute.

4.1.4.6 DRMAA_INPUT_PATH

The DRMAA_INPUT_PATH directive is used to represent the *drmaa_input_path* attribute.

4.1.4.7 DRMAA_JOB_CATEGORY

The DRMAA_JOB_CATEGORY directive is used to represent the *drmaa_job_category* attribute.

4.1.4.8 DRMAA_JOB_NAME

The DRMAA_JOB_NAME directive is used to represent the *drmaa_job_name* attribute.

4.1.4.9 DRMAA_JOIN_FILES

The DRMAA_JOIN_FILES directive is used to represent the *drmaa_join files* attribute.

4.1.4.10 DRMAA_JS_STATE

The DRMAA_JS_STATE directive is used to represent the *drmaa_js_state* attribute.

4.1.4.11 DRMAA_NATIVE_SPECIFICATION

The DRMAA_NATIVE_SPECIFICATION directive is used to represent the *drmaa_native_specification* attribute.

4.1.4.12 DRMAA_OUTPUT_PATH

The DRMAA_OUTPUT_PATH directive is used to represent the *drmaa_output_path* attribute.

4.1.4.13 DRMAA_REMOTE_COMMAND

The DRMAA_REMOTE_COMMAND directive is used to represent the *drmaa_remote_command* attribute.

4.1.4.14 DRMAA_START_TIME

The DRMAA_START_TIME directive is used to represent the *drmaa_start_time* attribute.

4.1.4.15 DRMAA_TRANSFER_FILES

The DRMAA_TRANSFER_FILES directive is used to represent the *drmaa_transfer_files* attribute.

4.1.4.16 DRMAA_V_ARGV

The DRMAA_V_ARGV directive is used to represent the *drmaa_v_argv* attribute.

4.1.4.17 DRMAA_V_EMAIL

The DRMAA_V_EMAIL directive is used to represent the *drmaa_v_email* attribute.

4.1.4.18 DRMAA_V_ENV

The DRMAA_V_ENV directive is used to represent the *drmaa_v_env* attribute.

4.1.4.19 DRMAA_WCT_HLIMIT

The DRMAA_WCT_HLIMIT directive is used to represent the *drmaa_wct_hlimit* attribute.

4.1.4.20 DRMAA_WCT_SLIMIT

The DRMAA_WCT_SLIMIT directive is used to represent the *drmaa_wct_slimit* attribute.

4.1.4.21 DRMAA_SUBMISSION_STATE_ACTIVE

The DRMAA_SUBMISSION_STATE_ACTIVE directive is used with the *drmaa_js_state* attribute to indicate that the job to be executed should not be placed into a user hold state.

4.1.4.22 DRMAA_SUBMISSION_STATE_HOLD

The DRMAA_SUBMISSION_STATE_HOLD directive is used with the *drmaa_js_state* attribute to indicate that the job to be executed should be immediately placed into a user hold state.

4.1.4.23 DRMAA_PLACEHOLDER_HD

The DRMAA_PLACEHOLDER_HD directive is used with the *drmaa_wd*, *drmaa_input_path*, *drmaa_output_path*, and *drmaa_error_path* attributes to represent the user's home directory.

4.1.4.24 DRMAA_PLACEHOLDER_WD

The DRMAA_PLACEHOLDER_WD directive is used with the *drmaa_input_path*, *drmaa_output_path*, and *drmaa_error_path* attributes to represent the job working directory.

4.1.4.25 DRMAA_PLACEHOLDER_INCR

The DRMAA_PLACEHOLDER_INCR directive is used with the *drmaa_wd*, *drmaa_input_path*, *drmaa_output_path*, and *drmaa_error_path* attributes to represent the individual id of each subjob in the parametric job. See drmaa_run_bulk_jobs().

4.1.5 C Preprocessor Directives for DRMAA Error Codes

A DRMAA header file SHALL define the following preprocessor directives:

```
#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
#define DRMAA_ERRNO_NO_DEFAULT_CONTACT_STRING_SELECTED 24
#define DRMAA_ERRNO_NO_MORE_ELEMENTS                   25
```

4.1.5.1 DRMAA_ERRNO_SUCCESS

The DRMAA_ERRNO_SUCCESS error code indicates that the called function returned normally with success.

4.1.5.2 DRMAA_ERRNO_INTERNAL_ERROR

The DRMAA_ERRNO_INTERNAL_ERROR error code indicates that an unexpected or internal DRMAA error, like system call failure, etc, has occurred.

4.1.5.3 DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE

The DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE error code indicates that the DRMAA implementation could not contact DRM system for this request.

4.1.5.4 DRMAA_ERRNO_AUTH_FAILURE

The DRMAA_ERRNO_AUTH_FAILURE error code indicates that the specified request was not processed successfully due to an authorization failure.

4.1.5.5 DRMAA_ERRNO_INVALID_ARGUMENT

The DRMAA_ERRNO_INVALID_ARGUMENT error code indicates that the value of an argument is invalid.

4.1.5.6 DRMAA_ERRNO_NO_ACTIVE_SESSION

The DRMAA_ERRNO_NO_ACTIVE_SESSION error code indicates that the function failed because there is no active session.

4.1.5.7 DRMAA_ERRNO_NO_MEMORY

The DRMAA_ERRNO_NO_MEMORY error code indicates that the system is unable to allocate the memory resources required to perform the requested operation.

4.1.5.8 DRMAA_ERRNO_INVALID_CONTACT_STRING

The DRMAA_ERRNO_INVALID_CONTACT_STRING error code indicates that session initialization failed due to an invalid contact string.

4.1.5.9 DRMAA_ERRNO_DEFAULT_CONTACT_STRING_ERROR

The DRMAA_ERRNO_DEFAULT_CONTACT_STRING_ERROR error code indicates that the DRMAA implementation could not use the default contact string to connect to any DRM system.

4.1.5.10 DRMAA_ERRNO_NO_DEFAULT_CONTACT_STRING_SELECTED

The DRMAA_ERRNO_NO_DEFAULT_CONTACT_STRING_SELECTED error code indicates that no default contact string was provided by or selected for drmaa_init(). DRMAA requires that the default contact string is selected when there is more than one default contact string available due to multiple DRMAA implementations contained in a single binary module.

4.1.5.11 DRMAA_ERRNO_DRMS_INIT_FAILED

The DRMAA_ERRNO_DRMS_INIT_FAILED error code indicates that session initialization failed because the DRMAA implementation was unable to initialize the DRM system.

4.1.5.12 DRMAA_ERRNO_ALREADY_ACTIVE_SESSION

The DRMAA_ERRNO_ALREADY_ACTIVE_SESSION error code indicates that session initialization failed due to an already existing DRMAA session.

4.1.5.13 DRMAA_ERRNO_DRMS_EXIT_ERROR

The DRMAA_ERRNO_DRMS_EXIT_ERROR error code indicates that disengagement from the DRM system failed.

4.1.5.14 DRMAA_ERRNO_INVALID_ATTRIBUTE_FORMAT

The DRMAA_ERRNO_INVALID_ATTRIBUTE_FORMAT error code indicates that the format of the job attribute value is invalid.

4.1.5.15 DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE

The DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE error code indicates that the value of the job attribute is invalid.

4.1.5.16 DRMAA_ERRNO_CONFLICTING_ATTRIBUTE_VALUES

The DRMAA_ERRNO_CONFLICTING_ATTRIBUTE_VALUES error code indicates that the value of this attribute conflicts with the value or values of one or more previously set attributes.

4.1.5.17 DRMAA_ERRNO_TRY_LATER

The DRMAA_ERRNO_TRY_LATER error code indicates that the DRMAA implementation could not perform the desired operation at this time, due to excessive load in the DRM system. A retry MAY succeed.

4.1.5.18 DRMAA_ERRNO_DENIED_BY_DRM

The DRMAA_ERRNO_DENIED_BY_DRM error code indicates that the DRM system rejected the job due to DRM configuration or job template settings. The job will never be accepted.

4.1.5.19 DRMAA_ERRNO_INVALID_JOB

The DRMAA_ERRNO_INVALID_JOB error code indicates that the specified job does not exist.

4.1.5.20 DRMAA_ERRNO_RESUME_INCONSISTENT_STATE

The DRMAA_ERRNO_RESUME_INCONSISTENT_STATE error code indicates that the job is not in a state from which it can be resumed, e.g. It is not in a user suspend state. The request SHALL NOT be processed.

4.1.5.21 DRMAA_ERRNO_SUSPEND_INCONSISTENT_STATE

The DRMAA_ERRNO_SUSPEND_INCONSISTENT_STATE error code indicates that the job is not in a state from which it can be suspended, e.g. It is not in a running state. The request SHALL NOT be processed.

4.1.5.22 DRMAA_ERRNO_HOLD_INCONSISTENT_STATE

The DRMAA_ERRNO_HOLD_INCONSISTENT_STATE error code indicates that the job is not in a state from which it can be held. The request SHALL NOT be processed.

4.1.5.23 DRMAA_ERRNO_RELEASE_INCONSISTENT_STATE

The DRMAA_ERRNO_RELEASE_INCONSISTENT_STATE error code indicates that the job is not in a state from which it can be released, e.g. It is not in a user hold state. The request SHALL NOT be processed.

4.1.5.24 DRMAA_ERRNO_EXIT_TIMEOUT

The DRMAA_ERRNO_EXIT_TIMEOUT error code indicates that the DRMAA implementation has encountered a time-out condition during a call to drmaa_synchronize() or drmaa_wait().

4.1.5.25 DRMAA_ERRNO_NO_RUSAGE

The DRMAA_ERRNO_NO_RUSAGE error code indicates that during a call to drmaa_wait(), the specified job has finished, but no rusage and/or stat data could be provided.

4.1.5.26 DRMAA_ERRNO_NO_DEFAULT_CONTACT_STRING_SELECTED

The DRMAA_ERRNO_NO_DEFAULT_CONTACT_STRING_SELECTED error code indicates that the call to drmaa_init() failed because more than one contact string is available from the DRMAA implementation, but NULL was passed as the *contact* parameter to drmaa_init().

4.1.5.27 DRMAA_ERRNO_NO_MORE_ELEMENTS

The DRMAA_ERRNO_NO_MORE_ELEMENTS error code indicates that the opaque string vector contains no further elements through which it is possible to iterate.

4.2 String Vector Helper Functions

A DRMAA C binding implementation SHALL provide the following functions for processing the opaque string vector data types:

```
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);
int drmaa_get_num_attr_names(drmaa_attr_names_t* values,
                             size_t *size);
int drmaa_get_num_attr_values(drmaa_attr_values_t* values,
                              size_t *size);
int drmaa_get_num_job_ids(drmaa_job_ids_t* values, size_t *size);
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);
```

Through these functions a program can iterate through the values contained in an opaque string vector once, moving from the first entry to the last entry. Once the opaque string vector has been iterated past the last entry, the opaque string vector data values are no longer accessible.

4.2.1 drmaa_get_next_attr_name

The drmaa_get_next_attr_name() function SHALL store up to *value_len* bytes of the next attribute name from the *values* opaque string vector in the *value* buffer. The opaque string vector's internal iterator SHALL then be moved forward to the next entry.

4.2.1.1 Parameters

`values` – The opaque string vector from which the next attribute name will be extracted.
`value` – The buffer into which the attribute name will be stored.
`value_len` – the length of the *value* buffer.

4.2.1.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.
`DRMAA_ERRNO_NO_MORE_ELEMENTS` – no more attribute names are available.

4.2.2 drmaa_get_next_attr_value

The drmaa_get_next_attr_value() function SHALL store up to *value_len* bytes of the next attribute value from the *values* opaque string vector in the *value* buffer. The opaque string vector's internal iterator SHALL then be moved forward to the next entry.

4.2.2.1 Parameters

`values` – The opaque string vector from which the next attribute value will be extracted.
`value` – The buffer into which the attribute value will be stored.
`value_len` – the length of the *value* buffer.

4.2.2.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.
`DRMAA_ERRNO_NO_MORE_ELEMENTS` – no more attribute value are available.

4.2.3 drmaa_get_next_job_id

The drmaa_get_next_job_id() function SHALL store up to *value_len* bytes of the next job id from the *values* opaque string vector in the *value* buffer. The opaque string vector's internal iterator SHALL then be moved forward to the next entry.

4.2.3.1 Parameters

`values` – The opaque string vector from which the next job id will be extracted.
`value` – The buffer into which the job id will be stored.
`value_len` – the length of the *value* buffer.

4.2.3.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.
`DRMAA_ERRNO_NO_MORE_ELEMENTS` – no more job ids are available.

4.2.4 drmaa_get_num_attr_names

The drmaa_get_num_attr_names() function SHALL store the number of elements in the space provided by *size*.

4.2.4.1 Parameters

`values` – The opaque string vector from whose number of elements will be returned.

`size` – Space into which to write the number of elements.

4.2.4.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.

4.2.5 drmaa_get_num_attr_values

The drmaa_get_num_attr_values() function SHALL store the number of elements in the space provided by *size*.

4.2.5.1 Parameters

`values` – The opaque string vector from whose number of elements will be returned.
`size` – Space into which to write the number of elements.

4.2.5.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.

4.2.6 drmaa_get_num_job_ids

The drmaa_get_num_job_ids() function SHALL store the number of elements in the space provided by *size*.

4.2.6.1 Parameters

`values` – The opaque string vector from whose number of elements will be returned.
`size` – Space into which to write the number of elements.

4.2.6.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.

4.2.7 drmaa_release_attr_names

The drmaa_release_attr_names() function frees the memory used by the *values* opaque string vector. All memory used by strings contained therein is also freed.

4.2.7.1 Parameters

`values` – The opaque string vector to be freed.

4.2.7.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – the argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.

4.2.8 drmaa_release_attr_values

The drmaa_release_attr_values() function frees the memory used by the *values* opaque string vector. All memory used by strings contained therein is also freed.

4.2.8.1 Parameters

`values` – The opaque string vector to be freed.

4.2.8.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – the argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.

4.2.9 drmaa_release_job_ids

The drmaa_release_attr_job_ids() function frees the memory used by the *values* opaque string vector. All memory used by strings contained therein is also freed.

4.2.9.1 Parameters

`values` – The opaque string vector to be freed.

4.2.9.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.

DRMAA_ERRNO_AUTH_FAILURE – the user is not authorized to perform this operation.
DRMAA_ERRNO_INVALID_ARGUMENT – the argument value is invalid.
DRMAA_ERRNO_NO_MEMORY – not enough free memory to perform the operation.
DRMAA_ERRNO_NO_ACTIVE_SESSION – no active session.

4.3 Session Management

A DRMAA C binding implementation SHALL provide the following functions for managing sessions:

```
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);
```

4.3.1 drmaa_init

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.

4.3.1.1 Parameters

contact – A string indicating to which DRMS the DRMAA session should bind during initialization.
error_diagnosis – A buffer into which error diagnosis information will be written.
error_diag_len – The size in characters of the error diagnosis string buffer.

4.3.1.2 Return Codes

DRMAA_ERRNO_SUCCESS – success.
DRMAA_ERRNO_INTERNAL_ERROR – unexpected or internal error.
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE – could not contact the DRMS for this request.
DRMAA_ERRNO_AUTH_FAILURE – the user is not authorized to perform this operation.
DRMAA_ERRNO_INVALID_ARGUMENT – an argument value is invalid.
DRMAA_ERRNO_NO_MEMORY – not enough free memory to perform the operation.
DRMAA_ERRNO_INVALID_CONTACT_STRING – the DRMAA implementation was unable to use the provided contact to contact the DRMS.
DRMAA_ERRNO_DEFAULT_CONTACT_STRING_ERROR – no contact string was provided, and the DRMAA implementation was unable to use the default contact to contact the DRMS.
DRMAA_ERRNO_NO_DEFAULT_CONTACT_STRING_SELECTED – there is no single default contact string, and no contact string was provided.
DRMAA_ERRNO_ERRNO_DRMS_INIT_FAILED – unable to initialize DRMS connection.
DRMAA_ERRNO_ERRNO_ALREADY_ACTIVE_SESSION – the session is already active.

4.3.2 drmaa_exit

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.

4.3.2.1 Parameters

`error_diagnosis` – A buffer into which error diagnosis information will be written.
`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.3.2.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.
`DRMAA_ERRNO_DRMS_EXIT_ERROR` – an error occurred while disconnecting from the DRMS.

4.4 Job Templates

A job template is a pattern used for defining job characteristics. Each job template is able to hold a different set of job characteristics. Upon job submission, the provided job template will be used to set the characteristics of the job to be submitted.

There is a 1:n relationship between job templates and jobs. A single job template can be used to submit any number of jobs. Once a job has been submitted, e.g. via drmaa_run_job(), the job template no longer has any affect on the job. Changes made to the job template will have no affect on already running jobs. Deleting the job template (via drmaa_delete_job_template()) also has no effect on running jobs.

A DRMAA C binding implementation SHALL provide the following functions for managing job templates:

```
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);
```

4.4.1 drmaa_allocate_job_template

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

4.4.1.1 Parameters

`jt` – The buffer to hold the newly allocated job template.
`error_diagnosis` – A buffer into which error diagnosis information will be written.
`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.4.1.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.

4.4.2 drmaa_delete_job_template

The function drmaa_delete_job_template() SHALL free the job template pointed to by *jt*.

4.4.2.1 Parameters

`jt` – The job template to be freed.
`error_diagnosis` – A buffer into which error diagnosis information will be written.

error_diag_len – The size in characters of the error diagnosis string buffer.

4.4.2.2 Return Codes

DRMAA_ERRNO_SUCCESS – success.
DRMAA_ERRNO_INTERNAL_ERROR – unexpected or internal error.
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE – could not contact the DRMS for this request.
DRMAA_ERRNO_AUTH_FAILURE – the user is not authorized to perform this operation.
DRMAA_ERRNO_INVALID_ARGUMENT – an argument value is invalid.
DRMAA_ERRNO_NO_MEMORY – not enough free memory to perform the operation.
DRMAA_ERRNO_NO_ACTIVE_SESSION – no active session.

4.4.3 drmaa_set_atrribute

The function drmaa_set_attribute() SHALL set the value of the scalar attribute, *name*, in the job template, *jt*, to the value, *value*.

4.4.3.1 Parameters

jt – The job template in which the attribute is to be set.
name – The name of the attribute to set.
value – The value to which to set the attribute.
error_diagnosis – A buffer into which error diagnosis information will be written.
error_diag_len – The size in characters of the error diagnosis string buffer.

4.4.3.2 Return Codes

DRMAA_ERRNO_SUCCESS – success.
DRMAA_ERRNO_INTERNAL_ERROR – unexpected or internal error.
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE – could not contact the DRMS for this request.
DRMAA_ERRNO_AUTH_FAILURE – the user is not authorized to perform this operation.
DRMAA_ERRNO_INVALID_ARGUMENT – an argument value is invalid.
DRMAA_ERRNO_NO_MEMORY – not enough free memory to perform the operation.
DRMAA_ERRNO_NO_ACTIVE_SESSION – no active session.
DRMAA_ERRNO_INVALID_ATTRIBUTE_FORMAT – the format of the attribute value is invalid for the attribute.
DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE – the attribute value is invalid.
DRMAA_ERRNO_CONFLICTING_ATTRIBUTE_VALUES – the attribute value conflicts with one or more prevously set attribute values.

4.4.4 drmaa_get_attribute

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.

4.4.4.1 Parameters

jt – The job template from which to get the attribute value.
name – The name of the attribute whose value will be retrieved.
value – A buffer into which the attribute's value will be written.
value_len – The size in characters of the attribute value buffer.
error_diagnosis – A buffer into which error diagnosis information will be written.

`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.4.4.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.

4.4.5 drmaa_set_vector_atrribute

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.

4.4.5.1 Parameters

`jt` – The job template in which the attribute is to be set.
`name` – The name of the attribute to set.
`value` – The value array to which to set the attribute.
`error_diagnosis` – A buffer into which error diagnosis information will be written.
`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.4.5.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.
`DRMAA_ERRNO_INVALID_ATTRIBUTE_FORMAT` – the format of one or more of the attribute values is invalid for the attribute.
`DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE` – one or more of the attribute value is invalid.
`DRMAA_ERRNO_CONFLICTING_ATTRIBUTE_VALUES` – one or more of the attribute value conflicts with one or more prevously set attribute values.

4.4.6 drmaa_get_vector_attribute

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.

4.4.6.1 Parameters

`jt` – The job template from which to get the attribute values.
`name` – The name of the attribute whose values will be retrieved.
`values` – An opaque string vector containing the attribute values
`error_diagnosis` – A buffer into which error diagnosis information will be written.

error_diag_len – The size in characters of the error diagnosis string buffer.

4.4.6.2 Return Codes

DRMAA_ERRNO_SUCCESS – success.
DRMAA_ERRNO_INTERNAL_ERROR – unexpected or internal error.
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE – could not contact the DRMS for this request.
DRMAA_ERRNO_AUTH_FAILURE – the user is not authorized to perform this operation.
DRMAA_ERRNO_INVALID_ARGUMENT – an argument value is invalid.
DRMAA_ERRNO_NO_MEMORY – not enough free memory to perform the operation.
DRMAA_ERRNO_NO_ACTIVE_SESSION – no active session.

4.4.7 drmaa_get_attribute_names

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.

4.4.7.1 Parameters

values – Space to write an opaque string vector containing the attribute names
error_diagnosis – A buffer into which error diagnosis information will be written.
error_diag_len – The size in characters of the error diagnosis string buffer.

4.4.7.2 Return Codes

DRMAA_ERRNO_SUCCESS – success.
DRMAA_ERRNO_INTERNAL_ERROR – unexpected or internal error.
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE – could not contact the DRMS for this request.
DRMAA_ERRNO_AUTH_FAILURE – the user is not authorized to perform this operation.
DRMAA_ERRNO_INVALID_ARGUMENT – an argument value is invalid.
DRMAA_ERRNO_NO_MEMORY – not enough free memory to perform the operation.
DRMAA_ERRNO_NO_ACTIVE_SESSION – no active session.

4.4.8 drmaa_get_vector_attribute_names

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.

4.4.8.1 Parameters

values – Space to write an opaque string vector containing the attribute names
error_diagnosis – A buffer into which error diagnosis information will be written.
error_diag_len – The size in characters of the error diagnosis string buffer.

4.4.8.2 Return Codes

DRMAA_ERRNO_SUCCESS – success.
DRMAA_ERRNO_INTERNAL_ERROR – unexpected or internal error.

DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE – could not contact the DRMS for this request.
DRMAA_ERRNO_AUTH_FAILURE – the user is not authorized to perform this operation.
DRMAA_ERRNO_INVALID_ARGUMENT – an argument value is invalid.
DRMAA_ERRNO_NO_MEMORY – not enough free memory to perform the operation.
DRMAA_ERRNO_NO_ACTIVE_SESSION – no active session.

4.4.9 Required Job Attributes

A DRMAA C binding implementation SHALL implement the following attributes. Vector attribute names contain a "_v_".

4.4.9.1 drmaa_remote_command

The *drmaa_remote_command* attribute specifies the remote command to execute. The *drmaa_remote_command* must be the path of an executable that is available at the job's execution host. If the path is relative, it is assumed to be relative to the working directory, usually set through the *drmaa_wd* attribute. If the working directory is not set, the path is interpreted in an implementation-specific manner. In any case, no binary file management is done.

4.4.9.2 drmaa_js_state

The *drmaa_js_state* attribute specifies the job's state at submission. The string values "drmaa_hold" and "drmaa_active" SHALL be supported. When "drmaa_active" is used, the job SHALL be submitted in a runnable state. When "drmaa_hold" is used, the job SHALL be submitted in a user hold state (either DRMAA_PS_USER_ON_HOLD or DRMAA_PS_USER_SYSTEM_ON_HOLD).

4.4.9.3 drmaa_wd

The *drmaa_wd* attribute specifies the directory name where the job will be executed. A "$drmaa_hd_ph$" placeholder at the beginning of the *drmaa_wd* value denotes the remaining string portion as a relative directory name which is resolved relative to the job user's home directory at the execution host. When the DRMAA job template is used for bulk job submission (See section 4.5.2), the "$drmaa_incr_ph$" placeholder can be used at any position within the *drmaa_wd* value to cause a substitution with the parametric job's index.

The *drmaa_wd* value must be specified in a syntax that is common at the host where the job is executed. If set to a relative path and no placeholder is used, the path is interpreted in an implementation-specific manner. If not set, the working directory will be set in an implementation-specific manner. If set and the given directory does not exist, the job will enter the DRMAA_PS_FAILED state when run.

4.4.9.4 drmaa_job_name

The *drmaa_job_name* attribute specifies the job's name. A job name SHALL contain only alpha-numeric and '_' characters.

4.4.9.5 drmaa_input_path

The *drmaa_input_path* attribute specifies the standard input path of the job. If set, this attribute's value specifies the network path of the job's input stream file. The value of the *drmaa_input_path* attribute must be of the form:

```
[hostname]:file_path
```

When the *drmaa_transfer_files* attribute is supported and contains the character “i”, the input file SHALL be fetched by the DRMS from the specified host or from the submit host if no hostname is specified in the *drmaa_input_path* attribute value. When the *drmaa_transfer_files* attribute is not supported or does not contain the character “i”, the input file is always expected at the host where the job is executed, regardless of any hostname specified in the *drmaa_input_path* attribute value.

If the DRMAA job template will be used for bulk job submission, (See section 4.5.2.) the “$drmaa_incr_ph$” placeholder can be used at any position within the *drmaa_input_path* attribute value to cause a substitution with the parametric job's index. A “$drmaa_hd_ph$” placeholder at the beginning of the *drmaa_input_path* attribute value denotes the remaining portion of the *drmaa_input_path* attribute value as a relative file specification resolved relative to the job submitter's home directory at the host where the file is located. A “$drmaa_wd_ph$” placeholder at the beginning of the *drmaa_input_path* attribute value denotes the remaining portion of the the *drmaa_input_path* attribute value as a relative file specification resolved relative to the job's working directory at the host where the file is located. The *drmaa_input_path* attribute value must be specified in a syntax that is common at the host where the file is located. If set and the file can't be read the job enters the state DRMAA_PS_FAILED upon submission.

4.4.9.6 drmaa_output_path

The *drmaa_output_path* attribute specifies the standard output path of the job. If set, this attribute's value specifies the network path of the job's output stream file. The value of the *drmaa_output_path* attribute must be of the form:

```
[hostname]:file_path
```

When the *drmaa_transfer_files* attribute is supported and contains the character, “o”, the output file SHALL be transferred by the DRMS to the specified host or to the submit host if no hostname is specified in the *drmaa_output_path* attribute value. When the *drmaa_transfer_files* attribute is not supported or does not contain the character, “o”, the output file is always kept at the host where the job is executed, regardless of any hostname specified in the *drmaa_output_path* attribute value.

If the DRMAA job template will be used for bulk job submission, (See section 4.5.2.) the “$drmaa_incr_ph$” placeholder can be used at any position within the *drmaa_output_path* attribute value to cause a substitution with the parametric job's index. A “$drmaa_hd_ph$” placeholder at the beginning of the *drmaa_output_path* attribute value denotes the remaining portion of the *drmaa_output_path* attribute value as a relative file specification resolved relative to the job submitter's home directory at the host where the file is located. A “$drmaa_wd_ph$” placeholder at the beginning of the *drmaa_output_path* attribute value denotes the remaining portion of the the *drmaa_output_path* attribute value as a relative file specification resolved relative to the job's working directory at the host where the file is located. The *drmaa_output_path* attribute value must be specified in a syntax that is common at the host where the file is located. If set and the file can't be written before execution the job enters the state DRMAA_PS_FAILED upon submission.

4.4.9.7 drmaa_error_path

The *drmaa_error_path* attribute specifies the standard error path of the job. If set, this attribute's value specifies the network path of the job's error stream file. The value of the *drmaa_error_path* attribute must be of the form:

```
[hostname]:file_path
```

When the *drmaa_transfer_files* attribute is supported and contains the character, “e”, the output file SHALL be transferred by the DRMS to the specified host or to the submit host if no hostname is specified in the *drmaa_error_path* attribute value. When the *drmaa_transfer_files* attribute is not supported or does not contain the character, “e”, the output file is always kept at the host where the job is executed, regardless of any hostname specified in the *drmaa_error_path* attribute value.

If the DRMAA job template will be used for bulk job submission, (See section 4.5.2.) the “$drmaa_incr_ph$” placeholder can be used at any position within the *drmaa_error_path* attribute value to cause a substitution with the parametric job's index. A “$drmaa_hd_ph$” placeholder at the beginning of the *drmaa_error_path* attribute value denotes the remaining portion of the *drmaa_error_path* attribute value as a relative file specification resolved relative to the job submitter's home directory at the host where the file is located. A “$drmaa_wd_ph$” placeholder at the beginning of the *drmaa_error_path* attribute value denotes the remaining portion of the the *drmaa_error_path* attribute value as a relative file specification resolved relative to the job's working directory at the host where the file is located. The *drmaa_error_path* attribute value must be specified in a syntax that is common at the host where the file is located. If set and the file can't be written before execution the job enters the state DRMAA_PS_FAILED upon submission.

4.4.9.8 drmaa_join_files

The *drmaa_join_files* attribute specifies whether the job's error stream should be intermixed with the job's output stream. If not explicitly set in the job template, the attribute's value defaults to “n”. Either “y” or “n” can be specified as valid attribute values. If “y” is specified as the *drmaa_join_files* attribute value, the DRMAA implementation will ignore the value of the *drmaa_error_path* attribute and intermix the standard error stream with the standard output stream at the location specified by the *drmaa_output_path* attribute value.

4.4.9.9 drmaa_v_argv

The *drmaa_v_argv* attribute specifies the array of string values which SHALL be passed as arguments to the job.

4.4.9.10 drmaa_job_category

The *drmaa_job_catrgory* attribute specifies the DRMAA job category. See section 2.4.1 of the Distributed Resource Management Application API Specification 1.0 for more information about DRMAA job categories.

4.4.9.11 drmaa_native_specification

The *drmaa_native_specification* attribute specifies the native submission options which SHALL be passed to the DRMS at job submission time. See section 2.4.2 of the Distributed Resource Management Application API Specification 1.0 for more information about DRMAA job categories.

4.4.9.12 drmaa_v_env

The *drmaa_v_env* attribute specifies the environment variable settings for the job to be submitted. Each value in the attribute's value array represent an environment variable setting of the form:

```
<name>=<value>
```

4.4.9.13 drmaa_v_email

The *drmaa_v_email* attribute specifies the list of e-mail addresses to which job completion and status reports are to be sent. Which reports are sent and when they are sent are determined by the DRMS configuration and the *drmaa_block_email* attribute value.

4.4.9.14 drmaa_block_email

The *drmaa_block_email* attribute specifies whether the sending of email SHALL blocked or not. If the DRMS configuration or the *drmaa_native_specification* or *drmaa_job_category* attribute would normally cause email to be sent in association with job events, the *drmaa_block_email* attribute value can will override that setting, causing no email to be sent.

If the attribute's value is “0”, no email will be sent, regardless of DRMS configuration or other attribute values. If the attribute's value “1”, the sending of email is unaffected.

4.4.9.15 drmaa_start_time

The *drmaa_start_time* attribute specifies the earliest point in time when the job may be eligible to be run. *drmaa_start_time* attribute value is of the format:

```
[[[[CC]YY/]MM/]DD] hh:mm[:ss] [{-|+}UU:uu]
```

where

- CC is the first two digits of the year [19,)
- YY is the last two digits of the year [00,99]
- MM is the two digits of the month [01,12]
- DD is the two digit day of the month [01,31]
- hh is the two digit hour of the day [00,23]
- mm is the two digit minute of the day [00,59]
- ss is the two digit second of the minute [00,61]
- UU is the two digit hours since (before) UTC [-11,12]
- uu is the two digit minutes since (before) UTC [00,59]

If the optional UTC-offset is not specified, the offset associated with the local timezone will be used. If any of the other optional fields are not specified, the time SHALL be resolved to the soonest time which can be constructed using the values of the specified fields, which is in the future at the time of resolution. That is to say that if the attribute's value is “10:00”, and it is resolved to a concrete time at 11:01am on November 24th, the time will be resolved to 10:00am on November 25th, because that is the soonest time which matches the specified fields and is in the future. If at 9:34am on December 1st the same time string is resolved again (such as by reusing the containing job template for another job submission), it will resolve to 10:00am on December 1st.

4.4.10 Optional Job Attributes

The following reserved attribute names are OPTIONAL in a conforming DRMAA implementation. For attributes that are implemented, the meanings are REQUIRED to be as follows.

Note that the list of attributes that are implemented may be programmatically obtained by using the drmaa_get_attribute_names() and drmaa_get_vector_attribute_names() functions.

4.4.10.1 drmaa_transfer_files

The *drmaa_transfer_files* attribute specifies, which of the standard I/O files (stdin, stdout and stderr) are to be transferred to/from the execution host. If not set, this attribute's value defaults to

"". The attribute's value may contain any of the characters, “e”, “i” and “o”. If the character, “e”, is present, the error stream will be transferred. If the character, “i”, is present, the input stream will be transferred. If the character, “o”, is present, the output stream will be transferred. See the *drmaa_input_path*, *drmaa_output_path* and *drmaa_error_path* for information about how to specify the standard input file, standard output file and standard error file and the effects of this attribute's value.

4.4.10.2 drmaa_deadline_time

The *drmaa_start_time* attribute specifies a deadline after which the DRMS will terminate a job.. *drmaa_start_time* attribute value is of the format:

```
[[[[CC]YY/]MM/]DD] hh:mm[:ss] [{-|+}UU:uu]
```

where

- CC is the first two digits of the year [19,)
- YY is the last two digits of the year [00,99]
- MM is the two digits of the month [01,12]
- DD is the two digit day of the month [01,31]
- hh is the two digit hour of the day [00,23]
- mm is the two digit minute of the day [00,59]
- ss is the two digit second of the minute [00,61]
- UU is the two digit hours since (before) UTC [-11,12]
- uu is the two digit minutes since (before) UTC [00,59]

If the optional UTC-offset is not specified, the offset associated with the local timezone will be used. If any of the other optional fields are not specified, the time SHALL be resolved to the soonest time which can be constructed using the values of the specified fields, which is in the future at the time of resolution. That is to say that if the attribute's value is “10:00”, and it is resolved to a concrete time at 11:01am on November 24th, the time will be resolved to 10:00am on November 25th, because that is the soonest time which matches the specified fields and is in the future. If at 9:34am on December 1st the same time string is resolved again (such as by reusing the containing job template for another job submission), it will resolve to 10:00am on December 1st.

4.4.10.3 drmaa_wct_hlimit

The *drmaa_wct_hlimit* attribute specifies how much wall clock time a job is allowed to consume before its limit has been exceeded. The DRMS SHALL terminate a job that has exceeded its wallclock time limit. Suspended time SHALL also be accumulated here.

This attribute's value MUST be of the form:

```
[[h:]m:]s
```

where

- h is one or more digits representing hours
- m is one or more digits representing minutes
- s is one or more digits representing seconds

4.4.10.4 drmaa_wct_slimit

The *drmaa_wct_slimit* attribute specifies an estimate as to how much wall clock time the job will need to complete. Suspended time SHALL also be accumulated here. This attribute is intended to assist the scheduler. If the time specified by this attribute's value in insufficient, the DRMAA implementation MAY impose a scheduling penalty.

This attribute's value MUST be of the form:

```
[[h:]m:]s
```

where

- h is one or more digits representing hours
- m is one or more digits representing minutes
- s is one or more digits representing seconds

4.4.10.5 drmaa_duration_hlimit

The *drmaa_duration_hlimit* attribute specifies how long the job MAY be in a running state before its time limit has been exceeded, and therefore is terminated by the DRMS.

This attribute's value MUST be of the form:

```
[[h:]m:]s
```

where

- h is one or more digits representing hours
- m is one or more digits representing minutes
- s is one or more digits representing seconds

4.4.10.6 drmaa_duration_slimit

The *drmaa_duration_slimit* attribute specifies an estimate as to how long the job will need to remain in a running state in order to complete. This attribute is intended to assist the scheduler. If the time specified by this attribute's value in insufficient, the DRMAA iimplementation MAY impose a scheduling penalty.

This attribute's value MUST be of the form:

```
[[h:]m:]s
```

where

- h is one or more digits representing hours
- m is one or more digits representing minutes
- s is one or more digits representing seconds

4.5 Job Submission

A DRMAA C binding implementation SHALL provide the following functions for submitting jobs to be run by the DRMS:

```
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);
```

4.5.1 drmaa_run_job

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*.

4.5.1.1 Parameters

`job_id` – A buffer into which the submitted job's job identifier will be written.
`job_id_len` – The size in characters of the job identifier buffer.
`jt` – The job template in which the attribute is to be set.
`error_diagnosis` – A buffer into which error diagnosis information will be written.
`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.5.1.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.
`DRMAA_ERRNO_TRY_LATER` – the DRMAA implementation could not perform the desired operation at this time.
`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.

4.5.2 drmaa_run_bulk_jobs

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 divisble 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. See section **Error! Reference source not found.** for more information on these functions.

4.5.2.1 Parameters

`job_ids` – Space to write an opaque string vector containing the submitted jobs' job identifiers.
`jt` – The job template in which the attribute is to be set.
`start` – The minimum parametric job index to be assigned.
`end` – The maximum parametric job index to be assigned.
`incr` – The difference between consecuative parameteric job indices.
`error_diagnosis` – A buffer into which error diagnosis information will be written.
`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.5.2.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.
`DRMAA_ERRNO_TRY_LATER` – the DRMAA implementation could not perform the desired operation at this time.
`DRMAA_ERRNO_DENIED_BY_DRM` – the DRM system rejected the jobs. The jobs will never be accepted due to DRM configuration or job template settings.

4.6 Job Status and Control

A DRMAA C binding implementation SHALL provide the following functions for monitoring and controlling jobs:

```
int drmaa_control(const char *job_id, 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);
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, drmaa_job_info_t **info,
               char *error_diagnosis, size_t error_diag_len);
```

A DRMAA implementation SHALL also provide the following functions for interpreting the drmaa_wait() job exit information structure:

```
int drmaa_wjobid(char *jobid, size_t jobid_len,
                 drmaa_job_info_t *info, char* error_diagnosis,
                 size_t error_diag_len);
int drmaa_wrusage(drmaa_attr_values_t **rusage,
                  drmaa_job_info_t *info, char* error_diagnosis,
                  size_t error_diag_len);
int drmaa_wifexited(int *exited, drmaa_job_info_t *info,
                    char *error_diagnosis, size_t error_diag_len);
int drmaa_wexitstatus(int *exit_status, drmaa_job_info_t *info,
```

```
                        char *error_diagnosis, size_t error_diag_len);
    int drmaa_wifsignaled(int *signaled, drmaa_job_info_t *info,
                        char *error_diagnosis, size_t error_diag_len);
    int drmaa_wtermsig(char *signal, size_t signal_len,
                      drmaa_job_info_t *info, char *error_diagnosis,
                      size_t error_diag_len);
    int drmaa_wcoredump(int *core_dumped, drmaa_job_info_t *info,
                       char *error_diagnosis, size_t error_diag_len);
    int drmaa_wifaborted(int *aborted, drmaa_job_info_t *info,
                        char *error_diagnosis, size_t error_diag_len);
    int drmaa_wreason(char *reason, size_t reason_len,
                      drmaa_job_info_t *info, char *error_diagnosis,
                      size_t error_diag_len);
```

A DRMAA implementation SHALL also provide the following function for releasing the job exit information structure:

```
    void drmaa_release_job_info(drmaa_job_info_t* info);
```

4.6.1 drmaa_control

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

See section **Error! Reference source not found.** for more details.

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. See section 2.6 of the Distributed Resource Management Application API Specification 1.0 for more information about DRMAA job states.

4.6.1.1 Parameters

`job_id` – The job identifier of the job to be acted upon.
`action` – The action to be performed.
`error_diagnosis` – A buffer into which error diagnosis information will be written.
`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.6.1.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.

DRMAA_ERRNO_AUTH_FAILURE – the user is not authorized to perform this operation.
DRMAA_ERRNO_INVALID_ARGUMENT – an argument value is invalid.
DRMAA_ERRNO_NO_MEMORY – not enough free memory to perform the operation.
DRMAA_ERRNO_NO_ACTIVE_SESSION – no active session.
DRMAA_ERRNO_INVALID_JOB – the job does not exist or has already been reaped.
DRMAA_ERRNO_HOLD_INCONSISTENT_STATE – the HOLD action could not be performed.
DRMAA_ERRNO_RELEASE_INCONSISTENT_STATE – the RELEASE action could not be performed.
DRMAA_ERRNO_RESUME_INCONSISTENT_STATE – the RESUME action could not be performed.
DRMAA_ERRNO_SUSPEND_INCONSISTENT_STATE – the SUSPEND action could not be performed.

4.6.2 drmaa_job_ps

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 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_USER_SYSTEM_SUSPENDED
- DRMAA_PS_DONE
- DRMAA_PS_FAILED

Terminated jobs have a status of DRMAA_PS_FAILED. See section 2.6 of the Distributed Resource Management Application API Specification 1.0 for more information about DRMAA job states.

4.6.2.1 Parameters

job_id – The job identifier of the job to be queried.
remote_ps – Space to write the job's status information.
error_diagnosis – A buffer into which error diagnosis information will be written.
error_diag_len – The size in characters of the error diagnosis string buffer.

4.6.2.2 Return Codes

DRMAA_ERRNO_SUCCESS – success.
DRMAA_ERRNO_INTERNAL_ERROR – unexpected or internal error.
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE – could not contact the DRMS for this request.
DRMAA_ERRNO_AUTH_FAILURE – the user is not authorized to perform this operation.
DRMAA_ERRNO_INVALID_ARGUMENT – an argument value is invalid.
DRMAA_ERRNO_NO_MEMORY – not enough free memory to perform the operation.
DRMAA_ERRNO_NO_ACTIVE_SESSION – no active session.
DRMAA_ERRNO_INVALID_JOB – the job does not exist or has already been reaped.

4.6.3 drmaa_synchronize

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_synchroniize() call. If the *dispose* parameter's value is 0, the data record SHALL be left for future access via the drmaa_wait() method.

4.6.3.1 Parameters

`job_ids` – The job identifiers of the jobs to synchronize against.
`timeout` – The number of seconds to remain blocked waiting for a result. This value is signed because the value, DRMAA_TIMEOUT_WAIT_FOREVER, is equal to -1.
`dispose` – Whether to dispose of the jobs' exit status information.
`error_diagnosis` – A buffer into which error diagnosis information will be written.
`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.6.3.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.
`DRMAA_ERRNO_INVALID_JOB` – the job does not exist or has already been reaped.
`DRMAA_ERRNO_EXIT_TIMEOUT` – the timeout period elapsed before a result was received.

4.6.4 drmaa_wait

The drmaa_wait() function SHALL wait for a job identified by *job_id* to finish execution or fail. If the special string, DRMAA_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 preivously 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() SHALL fill *info* with the waited job's id, a code that includes information about the conditions under which the job terminated, and an array of strings that describe the amount of resources consumed by the job. The drmaa_wait() function MAY also fill *info* with additional information to be used by the drmaa_wifexited(), drmaa_wexitstatus(), drmaa_wifsignaled(), drmaa_wtermsig(), drmaa_wcoredump(), drmaa_wifaborted(), and drmaa_wreason() functions, such as the name of the terminating signal or the reason the job aborted. The *info* parameter is further described below.

The drmaa_wait() function reaps job data records on a successful call, so any subsequent calls to drmaa_wait() SHALL 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() exits due to a timeout, DRMAA_ERRNO_EXIT_TIMEOUT SHALL be returned and no rusage information SHALL be 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 *info* parameter, set by a successful call to drmaa_wait(), is used to retrieve further input about the exit information of the waited job, identified by the job id retrieved through the drmaa_wjobid() function, through the following functions: drmaa_wifexited(), drmaa_wexitstatus(), drmaa_wifsignaled(), drmaa_wtermsig(), drmaa_wcoredump(), drmaa_wifaborted(), and drmaa_wreason(). The resource usage information, retrieved through the drmaa_wrusage() function, is represented by an array of <name>=<value> strings that describe the amount of resources consumed by the job. These resource usage strings are implementation defined.

4.6.4.1 Parameters

`job_id` – The job identifier of the job for which to wait.
`info` – Space to write the job exit information of the finished job.
`error_diagnosis` – A buffer into which error diagnosis information will be written.
`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.6.4.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.
`DRMAA_ERRNO_INVALID_JOB` – the job does not exist or has already been reaped.
`DRMAA_ERRNO_EXIT_TIMEOUT` – the timeout period elapsed before a result was received.

4.6.5 drmaa_wjobid

The drmaa_wjobid() function SHALL store in *jobid* up to *jobid_len* bytes of the name of the job whose exit information is contained in *info*.

4.6.5.1 Parameters

`jobid` – A buffer in which to write the name of the finished job.
`jobid_len` – The size in characters of the job name buffer.
`info` – The exit information structure of a finished job.
`error_diagnosis` – A buffer into which error diagnosis information will be written.
`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.6.5.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.

4.6.6 drmaa_wrusage

The drmaa_wrusage() function SHALL place into *rusage* an opaque string vector containing the resource usage strings for the job whose exit information is stored in *info*. The resource usage string SHALL be of the format: <name>=<value>. These resource usage strings SHALL be implementation defined.

The drmaa_get_next_attr_value(), drmaa_get_num_attr_values(), and drmaa_release_attr_values() functions can be used to operate on the values stored in *rusage*. The opaque string vector placed into *rusage* SHALL be a copy of the information stored in *info*. Consequently, freeing *rusage* with the drmaa_release_attr_values() will have no effect on *info*.

4.6.6.1 Parameters

`rusage` – Space to write an opaque string vector containing usage information for the finished job.
`info` – The exit information structure of a finished job.
`error_diagnosis` – A buffer into which error diagnosis information will be written.
`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.6.6.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.

4.6.7 drmaa_wifexited

The drmaa_wifexited() function SHALL evaluate into *exited* a non-zero value if *info* was returned for a job that termined normally, and the job's exit status can be retrieved using the drmaa_wexitstatus() function. The *exited* parameter is filled with zero if either the job terminated

abnormally and hence has no exit status, or if the job terminated normally but nevertheless has no exit status. In the former case, more information can be provided about the job by the drmaa_wifsignaled() and drmaa_wcoredump() functions. In the later case, drmaa_wifsignaled() and drmaa_wifaborted() will provide no additional information.

4.6.7.1 Parameters

`exited` – Space to write whether the job has an exit status available.
`info` – The exit information structure of a finished job.
`error_diagnosis` – A buffer into which error diagnosis information will be written.
`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.6.7.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.

4.6.8 drmaa_wexitstatus

The drmaa_wexitstatus() function evaluates into *exit_status* the exit code extracted from *info*, provided that a call to drmaa_wifexited() with this *info* would return a non-zero value.

4.6.8.1 Parameters

`exit_status` – Space to write job's exit status.
`info` – The exit information structure of a finished job.
`error_diagnosis` – A buffer into which error diagnosis information will be written.
`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.6.8.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.

4.6.9 drmaa_wifsignaled

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

4.6.9.1 Parameters

`signaled` – Space to write whether the job terminated on a signal.
`info` – The exit information structure of a finished job.
`error_diagnosis` – A buffer into which error diagnosis information will be written.
`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.6.9.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.

4.6.10 drmaa_wtermsig

The drmaa_wtermsig() function SHALL fill *signal* with up to *signal_len* characters of the signal name that caused the termination of the job, provided that a call to drmaa_wifsignaled() with this *info* would return a non-zero value. For signals declared by POSIX, the symbolic names are returned. For non-POSIX signals, the returned names are implementation dependent.

4.6.10.1 Parameters

`signal` – A buffer in which to write the name of the signal.
`signal_len` – The size in characters of the signal buffer.
`info` – The exit information structure of a finished job.
`error_diagnosis` – A buffer into which error diagnosis information will be written.
`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.6.10.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.

4.6.11 drmaa_wcoredump

The drmaa_wcoredump() function SHALL fill *core_dumped* with a non-zero value provided that a call to drmaa_wifsignaled() with this *info* would return a non-zero value, and a core image of the terminated job was created.

4.6.11.1 Parameters

`core_dump` – Space in which to write whether the job produce a core image.
`info` – The exit information structure of a finished job.
`error_diagnosis` – A buffer into which error diagnosis information will be written.
`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.6.11.2 Return Codes

DRMAA_ERRNO_SUCCESS – success.
DRMAA_ERRNO_INTERNAL_ERROR – unexpected or internal error.
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE – could not contact the DRMS for this request.
DRMAA_ERRNO_AUTH_FAILURE – the user is not authorized to perform this operation.
DRMAA_ERRNO_INVALID_ARGUMENT – an argument value is invalid.
DRMAA_ERRNO_NO_MEMORY – not enough free memory to perform the operation.
DRMAA_ERRNO_NO_ACTIVE_SESSION – no active session.

4.6.12 drmaa_wifaborted

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

4.6.12.1 Parameters

aborted – Space in which to write whether the job aborted before running.
info – The exit information structure of a finished job.
error_diagnosis – A buffer into which error diagnosis information will be written.
error_diag_len – The size in characters of the error diagnosis string buffer.

4.6.12.2 Return Codes

DRMAA_ERRNO_SUCCESS – success.
DRMAA_ERRNO_INTERNAL_ERROR – unexpected or internal error.
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE – could not contact the DRMS for this request.
DRMAA_ERRNO_AUTH_FAILURE – the user is not authorized to perform this operation.
DRMAA_ERRNO_INVALID_ARGUMENT – an argument value is invalid.
DRMAA_ERRNO_NO_MEMORY – not enough free memory to perform the operation.
DRMAA_ERRNO_NO_ACTIVE_SESSION – no active session.

4.6.13 drmaa_wreason

The drmaa_wreason() function SHALL fill *reason* with up to *reason_len* characters of the a string value describing the reason why the job ended before entering the running state, provided that a call to drmaa_wifaborted() with this *info* would return a non-zero value.

4.6.13.1 Parameters

reason – A buffer in which to write a string describing the reason the job aborted.
reason_len – The size in characters of the abort reason buffer.
info – The exit information structure of a finished job.
error_diagnosis – A buffer into which error diagnosis information will be written.
error_diag_len – The size in characters of the error diagnosis string buffer.

4.6.13.2 Return Codes

DRMAA_ERRNO_SUCCESS – success.
DRMAA_ERRNO_INTERNAL_ERROR – unexpected or internal error.
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE – could not contact the DRMS for this request.

DRMAA_ERRNO_AUTH_FAILURE – the user is not authorized to perform this operation.
DRMAA_ERRNO_INVALID_ARGUMENT – an argument value is invalid.
DRMAA_ERRNO_NO_MEMORY – not enough free memory to perform the operation.
DRMAA_ERRNO_NO_ACTIVE_SESSION – no active session.

4.6.14 drmaa_release_job_info

The drmaa_release_job_info() function SHALL free all memory associated with the job exit information structure, *info*, including all internal strings and structures.

4.6.14.1 Parameters

info – The exit information structure of a finished job.

4.6.14.2 Return Codes

DRMAA_ERRNO_SUCCESS – success.
DRMAA_ERRNO_INTERNAL_ERROR – unexpected or internal error.
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE – could not contact the DRMS for this request.
DRMAA_ERRNO_AUTH_FAILURE – the user is not authorized to perform this operation.
DRMAA_ERRNO_INVALID_ARGUMENT – the argument value is invalid.
DRMAA_ERRNO_NO_MEMORY – not enough free memory to perform the operation.
DRMAA_ERRNO_NO_ACTIVE_SESSION – no active session.

4.7 Auxilliary Functions

A DRMAA C binding implementation SHALL provide the following auxilliary functions:

```
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);
```

4.7.1 drmaa_strerror

The drmaa_strerror() function SHALL return the error string describing the DRMAA error number *drmaa_errno*.

4.7.1.1 Parameters

drmaa_errno – The error code for which a string description is to be returned.

4.7.1.2 Return Value

Upon success, the drmaa_strerror() function SHALL return a string description for the error code, *drmaa_errno*. If *drmaa_errno* is not a recognized error code, the drmaa_strerror() function SHALL return NULL.

4.7.2 drmaa_get_contact

The drmaa_get_contacts() 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.

4.7.2.1 Parameters

`contact` – A buffer into which the contact string(s) will be written.
`contact_len` – The size in characters of the contact string buffer.
`error_diagnosis` – A buffer into which error diagnosis information will be written.
`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.7.2.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.

4.7.3 drmaa_version

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.

4.7.3.1 Parameters

`major` – Space into which the major version number will be written.
`minor` – Space into which the minor version number will be written.
`error_diagnosis` – A buffer into which error diagnosis information will be written.
`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.7.3.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.
`DRMAA_ERRNO_NO_ACTIVE_SESSION` – no active session.

4.7.4 drmaa_get_DRM_system

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.

4.7.4.1 Parameters

`drm_system` – A buffer into which the DRM system identifier(s) will be written.
`drm_system_len` – The size in characters of the DRM system identifier buffer.
`error_diagnosis` – A buffer into which error diagnosis information will be written.
`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.7.4.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.

4.7.5 drmaa_get_DRMAA_implementation

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.

4.7.5.1 Parameters

`drmaa_impl` – A buffer into which the DRMAA implementation identifier(s) will be written.
`drmaa_impl_len` – The size in characters of the DRMAA implementation identifier buffer.
`error_diagnosis` – A buffer into which error diagnosis information will be written.
`error_diag_len` – The size in characters of the error diagnosis string buffer.

4.7.5.2 Return Codes

`DRMAA_ERRNO_SUCCESS` – success.
`DRMAA_ERRNO_INTERNAL_ERROR` – unexpected or internal error.
`DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE` – could not contact the DRMS for this request.
`DRMAA_ERRNO_AUTH_FAILURE` – the user is not authorized to perform this operation.
`DRMAA_ERRNO_INVALID_ARGUMENT` – an argument value is invalid.
`DRMAA_ERRNO_NO_MEMORY` – not enough free memory to perform the operation.

5. C binding example

The C 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 the 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 the value for the “drmaa_remote_command” job template attribute. The C binding example program below 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 the example program expects, in this case, 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 = NULL;
   drmaa_job_template_t *jt = NULL;

   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 = NULL;
      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 %s\n", diagnosis,
                 drmaa_strerror(drmaa_errno));
         sleep(1);
      }

      if (drmaa_errno != DRMAA_ERRNO_SUCCESS) {
         fprintf(stderr, "drmaa_run_bulk_jobs() failed: %s %s\n", diagnosis,
                 drmaa_strerror(drmaa_errno));
         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;
      }

      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;
}
```

6. Security Considerations

The DRMAA API does not specifically assume the existence of a 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 that 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. To address 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 DRMAA will most likely require a network to coordinate subordinate DRMS. However, the API makes no assumptions about the security posture provided by the networking environment. Therefore, application developers should further consider the security implications of the "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.

7. 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

Daniel Templeton
dan.templeton@sun.com
Sun Microsystems GmbH
Dr.-Leo-Ritter-Str. 7
D-93049 Regensburg
Germany

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

Peter Tröger
peter.troeger@hpi.uni-potsdam.de
Hasso-Plattner-Institute, University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam
Germany

8. Intellectual Property Statement

The GGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license 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.

9. Full Copyright Notice