GWD-R

Hrabri Rajic, Intel Americas Inc. (maintainer)

Distributed Resource Management
Application API (DRMAA) Working Group

Roger Brobst, Cadence Design Systems
Waiman Chan, IBM
Fritz Ferstl, Sun Microsystems
Jeff Gardiner, John P. Robarts Res. Inst.
Andreas Haas, Sun Microsystems
Bill Nitzberg, Veridian Systems
Hrabri Rajic*, Intel Americas Inc.
John Tollefsrud*, Sun Microsystems
*co-chairs

February, 2003

Distributed Resource Management Application API Specification 1.0

Abstract

This document describes the Distributed Resource Management Application API (DRMAA), which provides a generalized API to distributed resource management systems (DRMSs) in order to facilitate integration of application programs.

The scope of DRMAA is limited to job submission, job monitoring and control, and retrieval of the finished job status. DRMAA provides application developers and distributed resource management builders with a programming model that enables the development of distributed applications tightly coupled to an underlying DRMS. For deployers of such distributed applications, DRMAA preserves flexibility and choice in system design.

Table of Contents

1. Introduction

The objective of the Global Grid Forum (GGF) Distributed Resource Management Application API (DRMAA) Working Group is to develop an API specification for the submission and control of jobs to one or more distributed resource management systems (DRMS).

The specification is to encompass all the high-level functionality necessary for an application to consign a job to a DRMS, including common operations on jobs like termination or suspension. The objective is to facilitate the direct interfacing of applications to today's DRMS by application builders, portal builders, and independent software vendors.

This document describes DRMAA specification 1.0. The specification abstracts the fundamental job interfaces of DRMS and provides an easy-to-use programming model, thereby encouraging adoption by both application builders and DRMS builders.

1.1 DRMAA Scope

The scope of DRMAA is limited to job submission, job monitoring and control, and retrieval of the finished job status. Resource reservation, security, and so forth are topics that are beyond DRMAA scope. These topics are handled by other GGF working or research groups.

1.2 Language Issues

We maintain that the API should be implemented for multiple languages, C/C++ being a primary choice by virtue of its widespread use by independent software vendors. Additional choices are the scripting languages Perl and Python.

One can design an interface definition language (IDL) that will effectively resolve the issue of one interface serving multiple languages. Such an approach, however, would significantly slow progress on an implementation. DRMAA interfaces are described using an *IDL-like* language.

Another approach would be to design a protocol instead of the API. The DRMAA Working Group felt, however, that the DRMAA specification be useful in the *implementation* of such a protocol.

1.3 Library Issues

A complete library implementation would support all DRMS and have versions for either static or dynamic linkage. Such an implementation, however, was deemed neither feasible nor desirable. An alternative is support for multiple, but not all, DRMS. The packaging could come as one library, where a DRMS is selected at run time by setting an environment variable for the desired DRMS, or as one DRMS link per library. We advocate the latter approach, in which the shared library is selected at run time by the end users.

We expect that developers will be linking the library from serial and multithreaded codes. The library should be thread safe. A thread-safe DRMAA implementation allows a multithreaded application to use DRMAA interfaces without any explicit synchronization among the application threads. Before a multithreaded application can use any of DRMAA interfaces, however, the DRMAA initialization routine should be called by only one thread, probably the main thread. Similarly, the DRMAA library should be disengaged by only one thread.

Debugging of distributed programs is more challenging than debugging of single machine versions. This document advocates providing production and debugging or tracing versions of the DRMAA library.

The library should support extraction of the DRMAA API version number by external programs (such as SCCS's "what" and RCS's "ident") and should allow distributed applications to obtain this information programmatically.

1.4 User Program and DRMAA Interaction

All of the DRMS are asynchronous; they notify the end user of the status of a finished job via e-mail. Such notification is insufficient to satisfy the needs of the DRMAA library users. This document therefore proposes to deal with the asynchrony similar to Unix and Windows process interfaces, by blocking on the wait call for a specific job request or possibly for all of them in the same session. Support for reactive mode (such as used in the Globus Toolkit™ GRAM interface) is to be addressed in future DRMAA versions, to better support the programming model used by an increasing number of applications with graphical user interfaces

2. API Design Issues

Developers have been using Unix system, fork/exec, popen, and the wait interfaces for years to spawn additional processes and wait for the end of their execution to get their exit codes. Windows has equivalent utilities such as CreateProcess and WaitForSingleObject. DRMAA provides its own set of interfaces that are OS neutral. It attempts to be consistent with libc interfaces.

2.1 Basic Guidelines

Even though the API should be self-contained, it is not always possible to consolidate all variations of end user and DRMS interactions under the API. For this reason, this document advocates that developers provide a way for the end user to specify DRMS-specific options. The "job category" mechanism is intended to convey DRMS-specific options in a DRMS-independent fashion.

Additionally, one might need DRMS-specific options to be specified at run time as command line options. The end user could lose portability this way; but since most end users are concerned only with a single DRMS, this strategy is likely to be acceptable. In contrast, DRMAA providers and ISVs are much more likely to target multiple DRMSs and therefore require a DRMS-independent mechanism.

The API centers on the job_id attribute that is passed back by the DRMS upon job submission. Job_id is used for all the job control and monitoring purposes. An additional attribute, job_name, which is commonly found in DRMS implementations, is part of the job description. Job_name could be used by the developer or internally by the implementation to group the jobs for easier user classification and tracking. This attribute could be a key to achieve scalability for DRMAA implementations, especially since DRMS user jobs could be running concurrently with those of the other DRMS users.

The following guidelines were used in designing DRMAA:

- The API calling sequences should be simple and the API set small.

- The routine names should convey the semantics of the routine.
- The set should be as convenient as possible, even with the risk of being forced to emulate some functionality if missing from a DRMS.
- All job or job set manipulation should be available without explicit job iterating.
- The server names should be hidden, and the DRMS should be a “black box.”
- The end user should be able to specify native specification options attribute if he needs to interact with the DRMS beyond what is available through DRMAA job categories.
- The API should be extensible so that future implementations are backward compatible with earlier ones.
- Implementation-specific details will need to be documented by the DRMAA library providers.

2.2 Distributed Application Environment

The DRMAA interfaces and programming model focus on providing sufficient mechanisms for submitting a job, monitoring and controlling it, and obtaining its final status. Ideally, DRMAA implementations and distributed applications should not be concerned with a particular DRMS environment and DRMS site policies. Particularly for DRMAA implementations, however, this is not always possible to achieve, and therefore solutions with minimal intrusiveness, such as “job categories” and “native specification,” have been proposed. They abstract or aggregate the site-specific policies into simple strings that are interpreted by DRMAA implementations.

2.2.1 File Staging

DRMAA specification 1.0 does not have explicit file staging mechanisms. Setting implementation-specific job template attributes could enable file staging, provided the DRMAA implementation and the DRM system supports it.

2.2.2 Job Categories

The DRMAA interface specification should allow ISVs to write DRM-enabled applications even though the properties of a concrete DRMS installation, in particular the configuration of the DRMS, cannot be known in advance.

Experience with integrations based on DRM CLI show that even when the same ISV application is run as a job with the same DRMS, the site-specific policies in effect differ widely. These policies are typically concern site-specific attributes such as what resources are to be used by the job, preferences where to run the job, and how the job should be scheduled relative to other jobs.

For supporting the variety of policies, job-specific requests expressed by DRMS submit options are common in the DRMS product space. Usually, however, these options do not affect the job from the perspective of the ISV or of the end user. This observation is the basis for “job categories,” which insulate the ISV and end user from site-specific policies.

The “job categories” concept is the approach the DRMAA Working Group recommends for encapsulating site-specific details and completely hiding these details from applications using the DRMAA interface. The basic idea is to allow site administrators to create a job category that is suitable for an application to be dispatched by the DRMS; the associated category name will be specified as a job submission attribute.The category name can be used by the DRMAA library to determine site-specific resource and functional requirements of jobs in the category. Such requirements need to be configurable by the site operating a DRMS and deploying an ISV application on top of it.

An example can help to illustrate this idea:

> At site A, rendering application X is used in a heterogeneous clustered environment that is managed by a DRMS. Since application X is available only at a subset of these machines, the administrator sets up the DRMS so that the end users must put a -l X=true into their submit command line.
>
> At site B, the same application is used in a homogeneous clustered environment with rendering application X supported at all machines managed by the DRMS. However, since X jobs do compete with applications Y sharing the same resources and X applications are to be treated with higher priority than Y jobs, end users need to put a -p 1023 into their submit command line for raising the dispatch priority.
>
> An integration based on categories will allow submitting X jobs through the DRMAA interface in compliance with the policies of both sites A and B without the need to know about these policies. The ISV does this by specifying "X" as the category used for X rendering jobs submitted through the DRMAA interface and by mentioning this in the "DRMS integration" section of the X rendering software documentation.
>
> The administrators at site A and site B read the documentation or installation instructions about the "X" DRMAA category. The documentation of their DRMS contains directions about the category support of their DRMAA interface implementation. From this documentation they learn how to configure their DRMS in a way that "-l X=true" is used for "X" jobs at site A while "-p 1023" is used at site B for those jobs.

As far as the DRMAA interface specification is concerned, only a standardized mechanism for specifying the category is required. The mechanism for associating the policy-related portion of the submit command line to the job is to be delivered by each DRMAA implementation. A standardization of this mechanism is beyond the DRMAA standardization effort because it is too closely related to the administrative interface and it is anticipated that, for different DRMS, different mechanisms will be appropriate.

2.2.3 Native Specification

The categories concept provides a means for completely hiding site-specific policy details to be considered with a DRMAA job submission for a whole class of jobs. The drawback, however, is that the concept requires one job category to be maintained for each policy to be used.

In order to allow the DRMAA interface to also be used for the submission of jobs where job-individual policy specification is required "native specification" is supported. Native specification can be used without the requirement to maintain job categories. Instead of specifying a category name and having the DRMAA implementation associate the corresponding job submit options; native specification allows these submit options to be specified directly.

An example can help to illustrate this idea:

In order to implement the example from Section 2.2.2 via native specifications, the native option string "-l X=true" has to be passed directly to the DRMAA interface while "-p 1023" has to be used at site B.

As far as the DRMAA interface specification is concerned, the native specification is an opaque string and is interpreted by each DRMAA library. One can use job categories and native specification with the same job submission for policy specification. In this case, the DRMAA library is assumed to be capable of joining the outcome of the two policy sources in a reasonable way.

2.2.4 Building Portals

The nature of the DRMAA implementation, as a shared library, makes it a good candidate for inclusion in a Web server to support a Web portal to a DRMS.

DRMAA library could be

- linked by a collection of CGI scripts that are referenced by resident Web pages,
- linked in a Web server as a separate module, or
- built as a Perl (the same applies to a lesser degree to other scripting languages) module that is included in mod_perl or a module accessed from Perl CGI scripts.

Questions about maintaining state, security, and authentication and authorization require that the DRMAA implementation be viewed as just one component of a DRMS Web portal. Clearly, evaluation of the Web portal needs and their impact on DRMAA specification is beyond the scope of the current document and DRMAA charter.

2.3 Interface Routines General Description

The interface routines are grouped in five categories: init and exit, job template handling, job submission, job monitoring and control, and auxiliary or system routines such as error message routines.

2.3.1 Init and Exit Routines

The calling sequence of the init routine should allow all of the considered DRMS to be properly initialized, by interfacing either to the batch queue commands or to the DRMS API. Likewise, the exit routine should require parameters that will permit proper DRMS disengagement.

2.3.2 Job Template Routines

The remote jobs and their attributes are specified by the job template, which is an opaque parameter. The job attributes can be a string or a vector of string values.

The following job attributes are required:

- Remote command to execute
- Remote command input parameters, a vector parameter
- Job state at submission

- Job environment, a vector parameter
- Job working directory
- Job category
- Native specification
- Standard input, output, and error streams
- E-mail distribution list to report the job completion and status, a vector parameter
- E-mail suppression
- Job start time
- Job name to be used for the job submission

2.3.3 Job Submission Routines

The job submission routines come in two versions. There is one version for submitting individual jobs and one version for submitting bulk jobs.

2.3.4 Job Monitoring and Controlling Routines

The job monitoring and controlling API handles several functions:

- Job releasing, stopping, resuming, and killing
- Checking the exit code of the finished remote job
- Checking the remote job status
- Waiting for the remote job till the end of its execution
- Waiting for all the jobs or a subset of the current session jobs to finish execution (this is a useful synchronization mechanism)

The Unix and Windows signals are replaced with the job control routines that have counterparts in DRMS. The only nontraditional feature is the passing of DRMAA_JOB_IDS_SESSION_ALL string as job_id to indicate operations on all job_ids in the current process.

The remote job can be in following states:

- System hold
- User hold
- System and user hold simultaneously
- Queued active
- System suspended
- User suspended
- System and user suspended simultaneously
- running
- Finished (un)successfully

To this list we need to add the possibility that the DRMAA implementation may be unable to determine the status of the remote job. Note that a rejected job is not assigned a job_id and consequently cannot have a state.

2.3.5 Auxiliary Routines

The auxiliary routines are needed to obtain a textual representation of errors and other DRMAA implementation-specific information.

2.3.6 DRMAA Job State Transition Diagram

Figure 1 shows the DRMAA job state transition diagram in Harel notation:

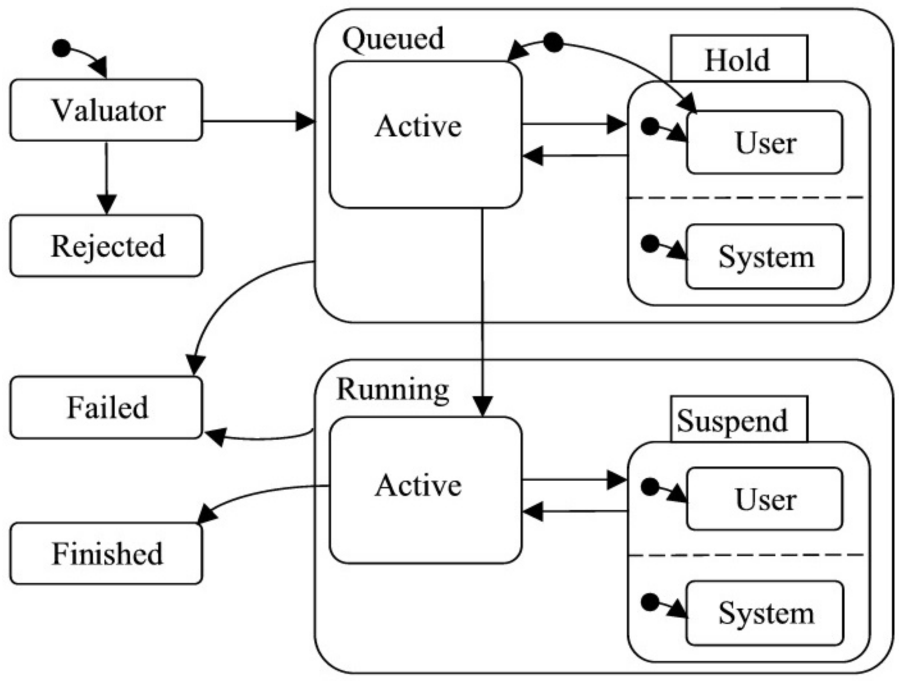

Figure 1: DRMAA job state transition diagram

3. API Specification

The API uses an IDL-like language to avoid questions about allocation and deallocation questions, which are specific to an implementation language. The interface parameters could be IN, OUT, or INOUT parameters. Readers who are familiar with the C programming language should think of the parameters as being passed by value or by reference. Furthermore, the parameters could be scalar or vector values. The vector values are clearly documented.

3.1 Routines

In order to prevent interface name collisions all the routines have a prefix “drmaa.”

3.1.1 Error Codes

All of the interfaces (except a few that return scalar values and cannot fail) return an error code on exit. Successful return is indicated by return value DRMAA_ERRNO_SUCCESS. All internal errors are indicated with DRMAA_ERRNO_INTERNAL_ERROR error. An invalid argument is flagged as DRMAA_ERRNO_INVALID_ARGUMENT. The return codes are specified and listed in Section 3.3.

The error code can be provided to the drmaa_strerror routine to retrieve a textual representation of the error. Routines may output a context-specific error string thatmay be used in addition to the textual representation obtained from the error code. The parameter used to convey the context specific error will be ignored by the routine whenever success is returned. The length of any output context-specific error string will not exceed DRMAA_ERROR_STRING_BUFFER.

3.1.2 DRMAA Sessions

Only one DRMAA session is open at the time. Another session can be opened only after the current one is closed. Nesting of sessions is not supported. It is expected that the DRMAA library will free all the session resources, although this is not guaranteed, so old session resources are not to be used later. Job Ids are valid from one session to another. Job control routines should work correctly if a job Id came from a previous DRMAA session, provided the current DRMAA session knows how to resolve this job Id. The burden is on the user to match previous job Id’s with appropriate DRMAA sessions (i.e., DRM system servers). Restartable applications have to make job Id’s persistent in order to access the already submitted jobs.

3.1.3 Run Usage Data

A DRMAA implementation collects remote run usage data (rusage variable) after the remote job run and job finish information (stat variable). The user can reap this data only once. The implementation is free to "garbage collect" the reaped data at a convenient time. Only the data from the current session's job Id is guaranteed to be available. Reaping data from other session job Id's is implementation specific.

3.1.4 Precedence Rules

The attributes set by using API routines are set at the compile time. The attributes set by job categories are set at installation time. The attributes set by the native specification are set at the run time. In principle these should determine the precedence rules, but these ideal precedence rules are not always achievable in practice because of complex interaction of attributes. Moreover, certain attributes in job categories may not be allowed to be overridden. The precedence rules are therefore implementation specific.

3.1.5 Site-Specific Requirements

Job categories and native specifications are two means for describing site specific requirements Setting job categories, if they are supported by a DRMAA implementation, is implementation specific. On the other hand, setting the native specification, while straightforward in the user code, could be a challenge if the user needs to provide a complex set of options. Quotation marks are especially problematic if only one variable is used for a set of native specification options.

The following recommendations are made to developers as to how to use this feature effectively:

? For each class of remote jobs, give end users a chance to specify site-specific environments, such as a queue where to send remote jobs or architecture(s) where the remote applications are available.

? Let users specify native specifications in a file if the distributed application has several classes of jobs to submit or several DRMAA sessions.

? Applications with a graphical user interface could have a dedicated dialog for this purpose.

3.1.6 Job Valuator

Before a submitted job enters a queue, it is passed through a valuator that determines whether the job attributes as specified are valid. If yes, a job Id is returned, and the job is successfully queued. If not, the job is rejected, no job Id is returned, and no job state is possible.

3.2 DRMAA API

For convenience, the API is divided in its five logical sections: init/exit, job template handling, job submission, job monitoring and control, and auxiliary routines.

```
/* ---------- Major Assumptions/Restrictions ---------- */

  No explicit file staging.
  Job Id Uniqueness -- "As unique as the underlying DRM makes them"


   /*Global constants */

       DRMAA_ERROR_STRING_BUFFER          = 1024
       DRMAA_JOBNAME_BUFFER               = 1024
       DRMAA_SIGNAL_BUFFER                =   32
       DRMAA_TIMEOUT_WAIT_FOREVER         =   -1
       DRMAA_TIMEOUT_NO_WAIT              =    0
```

3.2.1 Initialization and Exit Routines

drmaa_init(contact, drmaa_context_error_buf)

```
    IN  contact        /* contact information for DRM system (string) */
    INOUT drmaa_context_error_buf /*Contains a context-sensitive error upon failed return*/
```

Initialize DRMAA API library and create a new DRMAA session. 'Contact' is an implementation-dependent string that may be used to specify which DRM system to use. This routine must be called before any other DRMAA calls, except for drmaa_version().
If 'contact' is NULL, the default DRM system will be used. drmaa_init() should be called by only one of the threads. The main thread could be a good choice. A call by another thread would trigger a message that nested DRMAA sessions are not allowed.

drmaa_init routine returns DRMAA_ERRNO_SUCCESS on success, otherwise DRMAA_ERRNO_INVALID_CONTACT_STRING, DRMAA_ERRNO_ALREADY_ACTIVE_SESSION, or DRMAA_ERRNO_DEFAULT_CONTACT_STRING_ERROR.

drmaa_exit(drmaa_context_error_buf)
INOUT drmaa_context_error_buf /*Contains a context-sensitive error upon failed return*/

Disengage from DRMAA library and allow the DRMAA library to perform any necessary internal cleanup.
This routine ends this DRMAA session but does not affect any jobs (e.g., queued and running jobs remain queued and running). drmaa_exit() should be called by only one of the threads. Other thread calls to drmaa_exit() will fail since there is no active session.

drmaa_exit routine returns DRMAA_ERRNO_SUCCESS on success, otherwise DRMAA_ERRNO_DRMS_EXIT_ERROR or DRMAA_ERRNO_NO_ACTIVE_SESSION.

3.2.2 Job Template Routines

drmaa_allocate_job_template(jt, drmaa_context_error_buf)
jt /* job template (opaque handle) */
INOUT drmaa_context_error_buf /*Contains a context-sensitive error upon failed return*/

Allocate a new job template.
drmaa_allocate_job_template() returns DRMAA_ERRNO_SUCCESS on success, otherwise DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE or DRMAA_ERRNO_INTERNAL_ERROR.

drmaa_delete_job_template(jt, drmaa_context_error_buf)
INOUT jt /* job template (opaque handle) */
INOUT drmaa_context_error_buf /*Contains a context-sensitive error upon failed return*/

Deallocate a job template. This routine has no effect on jobs.
drmaa_delete_job_template() returns DRMAA_ERRNO_SUCCESS on success, otherwise DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE or DRMAA_ERRNO_INTERNAL_ERROR.

drmaa_set_attribute(jt, name, value, drmaa_context_error_buf)
INOUT jt /* job template (opaque handle) */

```
  IN    name                        /* attribute name (string) */
  IN    value                       /* attribute value (string) */
  INOUT drmaa_context_error_buf     /*Contains a context sensitive error upon failed return*/
```

Adds ('name', 'value') pair to list of attributes in job template 'jt'.
Only non-vector attributes may be passed.

drmaa_set_attribute routine returns DRMAA_ERRNO_SUCCESS on success, otherwise
DRMAA_ERRNO_INVALID_ATTRIBUTE_FORMAT,
DRMAA_ERRNO_INVALID_ARGUMENT,
DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE, or
DRMAA_ERRNO_CONFLICTING_ATTRIBUTE_VALUES.

drmaa_get_attribute(jt, name, value, drmaa_context_error_buf)

```
  IN    jt                          /* job template (opaque handle) */
  IN    name                        /* attribute name (string) */
  OUT   value                       /* attribute value (string) */
  INOUT drmaa_context_error_buf     /*Contains a context sensitive error upon failed return*/
```

If 'name' is an existing non-vector attribute name in the job template
'jt', then the value of 'name' is returned; otherwise, NULL is returned.

drmaa_get_attribute routine returns DRMAA_E RRNO_SUCCESS
on success, otherwise DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE.

drmaa_set_vector_attribute(jt, name, values, drmaa_context_error_buf)

```
  INOUT jt                          /* job template (opaque handle) */
  IN    name                        /* attribute name (string) */
  IN    values                      /* vector of attribute value (string vector) */
  INOUT drmaa_context_error_buf     /*Contains a context sensitive error upon failed return*/
```

Adds ('name', 'values') pair to list of vector attributes in job template 'jt'.
Only vector attributes may be passed.

drmaa_set_vector_attribute routine returns DRMAA_ERRNO_SUCCESS
on success, otherwise
DRMAA_ERRNO_INVALID_ATTRIBUTE_FORMAT,
DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE,
DRMAA_ERRNO_CONFLICTING_ATTRIBUTE_VALUES.

drmaa_get_vector_attribute(jt, name, values, drmaa_context_error_buf)

```
  IN    jt                          /* job template (opaque handle) */
  IN    name                        /* attribute name (string) */
  OUT   values                      /* vector of attribute value (string vector) */
```

If 'name' is an existing vector attribute name in the job template 'jt',
then the values of 'name' are returned; otherwise, NULL is returned.

drmaa_get_vector_attribute routine returns DRMAA_ERRNO_SUCCESS
on success, otherwise DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE.

```
drmaa_get_attribute_names( names, drmaa_context_error_buf )
   OUT names                        /* vector of attribute name (string vector) */
   INOUT drmaa_context_error_buf    /*Contains a context sensitive error upon failed return*/
```

Returns the set of supported attribute names whose associated value type is String. This set will include supported DRMAA reserved attribute names and native attribute names.

```
drmaa_get_vector_attribute_names( names, drmaa_context_error_buf )
   OUT names                        /* vector of attribute name (string vector) */
   INOUT drmaa_context_error_buf    /*Contains a context sensitive error upon failed return*/
```

Returns the set of supported attribute names whose associated value type is String Vector. This set will include supported DRMAA reserved attribute names and native attribute names.

3.2.3 Job attributes

3.2.3.1 Mandatory attributes

The following are reserved attribute names available in all implementations of DRMAA v1.0 (and their respective meanings). Vector attributes are marked with a 'V':

remote command to execute (string)
It is relative to the execution host.
It is evaluated on the execution host.
No binary file management is done.
The attribute name is drmaa_remote_command.

V input parameters (vector of strings)
These parameters are passed as arguments to the job.
The attribute name is drmaa_v_argv.

job state at submission (string value)
This might be useful for a rather rudimentary, but very general job-dependent execution.
The states are drmaa_hold and drmaa_active:
drmaa_active means job has been queued, and is eligible to run
drmaa_hold   means job has been queued, but it is NOT eligible to run
The attribute name is drmaa_js_state.

V job environment (vector of strings)
The environment values that define the remote environment.
Each string will comply with the format <name>=<value>.
The values override the remote environment values if there is a collision.
If above is not possible, it is implementation dependent.
The attribute name is drmaa_v_env.

job working directory (string)

This attribute specifies the directory where the job is executed.
If not set, it is implementation dependent.
Evaluated relative to the execution host.

A $drmaa_hd_ph$ placeholder at the begin denotes the remaining portion of the directory_name as a relative directory name resolved relative to the job users home directory at the execution host.
The $drmaa_incr_ph$ placeholder can be used at any position within the directory_name of parametric job templates and will be substituted by the underlying DRM system with the parametric jobs' index.
The directory_name must be specified in a syntax that is common at the host where the job is executed.
If set and no placeholder is used, an absolute directory specification is expected.
If set and the directory does not exist, the job enters the state DRMAA_PS_FAILED.
The attribute name is drmaa_wd.

job category (string)

An opaque string specifying how to resolve site-specific resources and/or policies.
The attribute name is drmaa_job_category.

native specification (string)

An opaque string that is passed by the end user to DRMAA to specify site-specific resources and/or policies.
The attribute name is drmaa_native_specification.

e-mail address (vector of strings)

It is used to report the job completion and status.
The attribute name is drmaa_v_email.

e-mail suppression (string)

It is used to block sending e-mail by default.
1 block
0 do not block.
The attribute name is drmaa_block_email

job start time (string)

This attribute specifies the earliest time when the job may be eligible to be run.
This is a required attribute named drmaa_start_time
The value of the attribute will be of the form

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

where

CC is the first two digits of the year (century-1)
YY is the last two digits of the year
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
uu is the two-digit minutes since (before) UTC

If the optional UTC-offset is not specified, the offset associated with the local timezone will be used.
If the day (DD) is not specified, the current day will be used unless the specified hour:mm:ss has already elapsed, in which case the next day will be used.
Similarly for month (MM), year (YY), and century-1 (CC).

Example:
The time: Sep 3 4:47:27 PM PDT 2002,
could be represented as: 2002/09/03 16:47:27 -07:00

job name

A job name will comprise alphanumeric and _ characters.
The drmaa-implementation will not provide the client with a job name longer than DRMAA_JOBNAME_BUFFER -1 (1023) characters.
The drmaa-implementation may truncate any client-provided job name to an implementation-defined length that is at least 31 characters.
The attribute name is drmaa_job_name

input stream (string)

Specifies the jobs' standard input.
Unless set elsewhere, if not explicitly set in the job template, the job is started with an empty input stream.
If set, specifies the network path of the jobs input stream file of the form [hostname]:file_path
When the drmaa_transfer_files job template attribute is supported and contains the character 'i', the input file will be fetched by the underlying DRM system from the specified host or from the submit host if no hostname is specified.
When the drmaa_transfer_files job template 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 irrespectively of a possibly hostname specified.
The $drmaa_incr_ph$ placeholder can be used at any position within the file_path of parametric job templates and will be substituted by the underlying DRM system with the parametric jobs' index.
A $drmaa_hd_ph$ placeholder at the begin of the file_path denotes the remaining portion of the file_path as a relative file specification resolved relative to the job users home directory at the host where the file is located.
A $drmaa_wd_ph$ placeholder at the begin of the file_path denotes the remaining portion of the file_path as a relative file specification resolved relative to the jobs working directory at the host where the file is located.
The file_path 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.
The attribute name is drmaa_input_path.

output stream (string)

Specifies how to direct the jobs' standard output.
If not explicitly set in the job template, the whereabouts of the jobs output stream is not defined.
If set, specifies the network path of the jobs output stream file of the form

[hostname]:file_path

When the drmaa_transfer_files job template attribute is supported and contains the character 'o', the output file will be transferred by the underlying DRM system to the specified host or to the submit host if no hostname is specified.

When the drmaa_transfer_files job template 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 irrespectively of a possibly hostname specified.

The $drmaa_incr_ph$ placeholder can be used at any position within the file_path of parametric job templates and will be substituted by the underlying DRM system with the parametric jobs' index.

A $drmaa_hd_ph$ placeholder at the begin of the file_path denotes the remaining portion of the file_path as a relative file specification resolved relative to the job users home directory at the host where the file is located.

A $drmaa_wd_ph$ placeholder at the begin of the file_path denotes the remaining portion of the file_path as a relative file specification resolved relative to the jobs working directory at the host where the file is located.

The file_path 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.

The attribute name is drmaa_output_path.

error stream (string)

Specifies how to direct the jobs’ standard error.

If not explicitly set in the job template, the whereabouts of the jobs error stream is not defined.

If set, specifies the network path of the jobs error stream file of the form [hostname]:file_path

When the drmaa_transfer_files job template attribute is supported and contains the character 'e', the output file will be transferred by the underlying DRM system to the specified host or to the submit host if no hostname is specified.

When the drmaa_transfer_files job template attribute is not supported or does not contain the character 'e', the error file is always kept at the host where the job is executed irrespectively of a possibly hostname specified.

The $drmaa_incr_ph$ placeholder can be used at any position within the file_path of parametric job templates and will be substituted by the underlying DRM system with the parametric jobs' index.

A $drmaa_hd_ph$ placeholder at the begin of the file_path denotes the remaining portion of the file_path as a relative file specification resolved relative to the job users home directory at the host where the file is located.

A $drmaa_wd_ph$ placeholder at the begin of the file_path denotes the remaining portion of the file_path as a relative file specification resolved relative to the jobs working directory at the host where the file is located.

The file_path 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.

The attribute name is drmaa_error_path.

join files (string)

Specifies if the error stream should be intermixed with the output stream.

If not explicitly set in the job template the attribute defaults to 'n'.

Either 'y' or 'n' can be specified.

If 'y' is specified the underlying DRM system will ignore the value of the drmaa_error_path attribute and intermix the standard error stream with the standard output stream as specified with drmaa_output_path.

The attribute name is drmaa_join_files.

3.2.3.2 Optional Attributes

The following are reserved attribute names available that are not required to be implemented by a conforming DRMAA v1.0 implementation. For attributes that are implemented, the meanings are required to be as follows:

Note that the list of attributes that are implemented can be programmatically obtained by using the drmaa_get_attribute_names and drmaa_get_vector_attribute_names routines.

transfer files (string)
Specifies how to transfer files between hosts.
If not explicitly set in the job template the attribute defaults to ''.
Any combination of 'e', 'i' and 'o' can be specified.
Whether the character 'e' is specified impacts the behavior of the drmaa_error_path attribute.
Whether the character 'i' is specified impacts the behavior of the drmaa_input_path attribute.
Whether the character 'o' is specified impacts the behavior of the drmaa_output_path attribute.
The attribute name is drmaa_transfer_files.

absolute job termination time (string)
This attribute specifies a deadline after which the DRMS will terminate a job.
This is a reserved attribute named drmaa_deadline_time
The value of the attribute will be of the form
[[[[CC]YY/]MM/]DD] hh:mm[:ss] [{-|+}UU:uu]
where
CC is the first two digits of the year (century-1)
YY is the last two digits of the year
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
uu is the two digit minutes since (before) UTC
If an optional portion of the time specification is omitted, then the termination time will be determined based upon the the job's earliest start time.
If the day (DD) is not specified, the earliest start day for the job will be used unless the specified hour:mm:ss precedes the corresponding portion of the job start time, in which case the next day will be used.

Similarly for month (MM), year (YY), and century-1 (CC).

Example:
The time: Sep 3 4:47:27 PM PDT 2002,
could be represented as: 2002/09/03 16:47:27 -07:00

wall clock time limit (string)

This attribute specifies when the job's wall clock time limit has been exceeded. The DRMS will terminate a job that has exceeded its wall clock time limit. Note that the suspended time is also accumulated here.

This is a reserved attribute named drmaa_wct_hlimit

The value of the attribute will 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

Example:
   To terminate a job after 2 hours and 30 minutes,
   any of the following can be passed:
       2:30:0, 1:90:0, 150:0
```

soft wall clock time limit (string)

This attribute specifies an estimate as to how long the job will need wall clock time to complete. Note that the suspended time is also accumulated here.

This attribute is intended to assist the scheduler. If the time specified in insufficient, the drmaa-implementation may impose a scheduling penalty.

This is a reserved attribute named drmaa_wct_slimit

The value of the attribute will 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
```

job run duration hlimit (string)

This attribute specifies how long the job may be in a running state before its limit has been exceeded, and therefore is terminated by the DRMS.

This is a reserved attribute named drmaa_run_duration_hlimit

The value of the attribute will 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
```

job run duration slimit (string)

This attribute specifies an estimate as to how long the job will need to remain in a running state to complete.

This attribute is intended to assist the scheduler. If the time specified in insufficient, the drmaa-implementation may impose a scheduling penalty.

This is a reserved attribute named drmaa_run_duration_slimit

The value of the attribute will 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
```

3.2.4 Job Submission Routines

drmaa_run_job(job_id, jt, drmaa_context_error_buf)

```
  OUT job_id                   /* job identifier (string) */
  IN  jt                       /* job template (opaque handle) */
  INOUT drmaa_context_error_buf /*Contains a context sensitive error upon failed return*/
```

Submit a job with attributes defined in the job template 'jt'.
The job identifier 'job_id' is a printable, NULL terminated string,
identical to that returned by the underlying DRM system.

drmaa_run_job routine returns DRMAA_ERRNO_SUCCESS on success,
otherwise DRMAA_ERRNO_TRY_LATER,
DRMAA_ERRNO_DENIED_BY_DRM,
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE, or
DRMAA_ERRNO_AUTH_FAILURE.

drmaa_run_bulk_jobs(job_ids, jt, start, end, incr, drmaa_context_error_buf)

```
  OUT job_ids                  /* job identifiers (array of strings) */
  IN  jt                       /* job template (opaque handle) */
  IN  start                    /* beginning index ( unsigned integer?)*/
  IN  end                      /* ending index ( unsigned integer?) */
  IN  incr                     /* loop increment (integer)*/
  INOUT drmaa_context_error_buf /*Contains a context sensitive error upon failed return*/
```

Submit a set of parametric jobs, dependent on the implied loop index, each
with attributes defined in the job template 'jt'.
The job identifiers 'job_ids' are all printable,
NULL terminated strings, identical to those returned by the underlying
DRM system. Nonnegative loop bounds are mandated to avoid file names
that start with minus sign like command line options.

The special index placeholder is a DRMAA defined string

```
  drmaa_incr_ph /* == $incr_pl$ */
```

this is used to construct parametric job templates.

For example:

drmaa_set_attribute(pjt, "stderr", drmaa_incr_ph + ".err"); /* C++/java string syntax used */

drmaa_run_bulk_jobs routine returns DRMAA_ERRNO_SUCCESS on success, otherwise DRMAA_ERRNO_TRY_LATER, DRMAA_ERRNO_DENIED_BY_DRM, DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE, or DRMAA_ERRNO_AUTH_FAILURE.

3.2.5 Job Control Routines

drmaa_control(job_id, action, drmaa_context_error_buf)

```
  IN job_id                      /* job identifier (string) */
  IN action                      /* control action (const) */
  INOUT drmaa_context_error_buf  /*Contains a context sensitive error upon failed return*/
```

Start, stop, restart, or kill the job identified by 'job_id'. If 'job_id' is DRMAA_JOB_IDS_SESSION_ALL, then this routine acts on all jobs *submitted* during this DRMAA session. The legal values for 'action' and their meanings are:

DRMAA_CONTROL_SUSPEND:	stop the job,
DRMAA_CONTROL_RESUME:	(re)start the job,
DRMAA_CONTROL_HOLD:	put the job on-hold,
DRMAA_CONTROL_RELEASE:	release the hold on the job, and
DRMAA_CONTROL_TERMINATE:	kill the job.

This routine returns once the action has been acknowledged by the DRM system, but does not necessarily wait until the action has been completed.

drmaa_control routine returns DRMAA_ERRNO_SUCCESS on success, otherwise DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE, DRMAA_ERRNO_AUTH_FAILURE, DRMAA_ERRNO_RESUME_INCONSISTENT_STATE DRMAA_ERRNO_SUSPEND_INCONSISTENT_STATE DRMAA_ERRNO_HOLD_INCONSISTENT_STATE DRMAA_ERRNO_RELEASE_INCONSISTENT_STATE DRMAA_ERRNO_INVALID_JOB.

drmaa_synchronize(job_ids, timeout, dispose, drmaa_context_error_buf)

```
  IN  job_ids                    /* job identifiers (array of strings) */
  IN  timeout                    /* how long we block in this call (signed long) */
  IN  dispose                    /* dispose reaping information (boolean)*/
  INOUT drmaa_context_error_buf  /*Contains a context sensitive error upon failed return*/
```

Wait until all jobs specified by 'job_ids' have finished execution. If 'job_ids' is DRMAA_JOB_IDS_SESSION_ALL, then this routine waits for all jobs *submitted* during this DRMAA session. To prevent blocking indefinitely in this call, the caller could use timeout specifying after how many seconds to time out in this call.

If the call exits before timeout, all the jobs have been waited on or there was an interrupt.

If the invocation exits on timeout, the return code is DRMAA_ERRNO_EXIT_TIMEOUT.
The caller should check system time before and after this call
in order to check how much time has passed.

Dispose parameter specifies how to treat reaping information:

True	= 1	"fake reap", i.e. dispose of the rusage data
False	= 0	do not reap

drmaa_ synchronize routine returns DRMAA_ERRNO_SUCCESS on success, otherwise
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE,
DRMAA_ERRNO_AUTH_FAILURE,
DRMAA_ERRNO_EXIT_TIMEOUT, or
DRMAA_ERRNO_INVALID_JOB.

drmaa_wait(job_id, stat, timeout, rusage, drmaa_context_error_buf)

```
   IN  job_id                          /* job identifier (string) or
                                       DRMAA_JOB_IDS_SESSION_ANY (string) */
   OUT job_id                          /* job identifier of ended job (string) or NULL */
   OUT stat                            /* status code of job (integer) */
   IN  timeout                         / * how long we block in this call (signed long) */
   OUT rusage                          /* resource usage (string array) */
   INOUT drmaa_context_error_buf  /*Contains a context sensitive error upon failed return*/
```

This routine waits for a job with job_id to fail or finish execution. If the special string
DRMAA_JOB_IDS_SESSION_ANY is provided as the job_id, this routine will wait for
any job from the session. This routine is modeled on the wait3 POSIX routine.
The timeout value is used to specify the desired behavior when a result is not
immediately available.
The value DRMAA_TIMEOUT_WAIT_FOREVER (-1) can be specified to wait
indefinitely for a result. The value DRMAA_TIMEOUT_NO_WAIT (0) may be specified
to return immediately if no result is available. Alternatively, a number of seconds may
be specified to indicate how long to wait for a result to become available.

If the call exits before timeout, either the job has been waited on
successfully or there was an interrupt.

If the invocation exits on timeout, the return code is DRMAA_ERRNO_EXIT_TIMEOUT.
The caller should check system time before and after this call
in order to check how much time has passed.

The routine reaps jobs on a successful call, so any subsequent calls
to drmaa_wait should fail returning an error DRMAA_ERRNO_INVALID_JOB meaning
that the job has been already reaped. This error is the same as if the job was
unknown. Failing due to an elapsed timeout has an effect that it is possible to
issue drmaa_wait multiple times for the same job_id. When successful, the rusage
information will be provided as an array of strings, where each string complies with the
format <name>=<value>.
The string portion <value> contains the amount of resources consumed
by the job and is opaque.

drmaa_ wait routine returns DRMAA_ERRNO_SUCCESS on success, otherwise

DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE,
DRMAA_ERRNO_AUTH_FAILURE,
DRMAA_ERRNO_NO_RUSAGE,
DRMAA_ERRNO_EXIT_TIMEOUT,
DRMAA_ERRNO_NO_RUSAGE, or
DRMAA_ERRNO_INVALID_JOB.

The 'stat' drmaa_wait parameter is used in a series of functions, defined below, for providing more detailed information about job termination if available. An analogous set of macros is defined in POSIX for analyzing wait3(2) OUT parameter 'stat'. The misleading upper-case function names reminding to macros are changed to lower-case names.

drmaa_wifexited(OUT exited, IN stat, INOUT drmaa_context_error_buf)
Evaluates into 'exited' a non-zero value if stat was returned for a job that terminated normally. A zero value can also indicate that although the job has terminated normally an exit status is not available or that it is not known whether the job terminated normally. In both cases drmaa_wexitstatus() will not provide exit status information. A non-zero 'exited' value indicates more detailed diagnosis can be provided by means of drmaa_wifsignaled(), drmaa_wtermsig() and drmaa_wcoredump().

drmaa_wexitstatus(OUT exit_status, IN stat, INOUT drmaa_context_error_buf)
If the OUT parameter 'exited' of drmaa_wifexited() is non-zero, this function evaluates into 'exit_code' the exit code that the job passed to _exit() (see exit(2)) or exit(3C), or the value that the child process returned from main.

drmaa_wifsignaled(OUT signaled, IN stat, INOUT drmaa_context_error_buf)
Evaluates into 'signaled' a non-zero value if status was returned for a job that terminated due to the receipt of a signal. A zero value can also indicate that although the job has terminated due to the receipt of a signal the signal is not available or that it is not known whether the job terminated due to the receipt of a signal. In both cases drmaa_wtermsig() will not provide signal information.

drmaa_wtermsig(OUT signal, IN stat, INOUT drmaa_context_error_buf)
If the OUT parameter 'signaled' of drmaa_wifsignaled(stat) is non-zero, this function evaluates into signal a string representation of the signal that caused the termination of the job. For signals declared by POSIX, the symbolic names are returned (e.g., SIGABRT, SIGALRM).
For signals not declared by POSIX, any other string may be returned.

drmaa_wcoredump(OUT core_dumped, IN stat, INOUT drmaa_context_error_buf)
If the OUT parameter 'signaled' of drmaa_wifsignaled(stat) is non-zero, this function evaluates into 'core_dumped' a non-zero value if a core image of the terminated job was created.

drmaa_wifaborted(OUT aborted, IN stat, INOUT drmaa_context_error_buf)
Evaluates into 'aborted' a non-zero value if 'stat'
was returned for a job that ended before entering the running state.

drmaa_job_ps(IN job_id, OUT remote_ps, drmaa_context_error_buf);

IN job_id	/* job identifier (string) */
OUT remote_ps	/* program status (constant) */
INOUT drmaa_context_error_buf	/*Contains a context sensitive error upon failed return*/

Get the program status of the job identified by 'job_id'.
The possible values returned in 'remote_ps' and their meanings are:

DRMAA_PS_UNDETERMINED	= 00H	: process status cannot be determined
DRMAA_PS_QUEUED_ACTIVE	= 10H	: job is queued and active
DRMAA_PS_SYSTEM_ON_HOLD	= 11H	: job is queued and in system hold
DRMAA_PS_USER_ON_HOLD	= 12H	: job is queued and in user hold
DRMAA_PS_USER_SYSTEM_ON_HOLD	= 13H	: job is queued and in user and system hold
DRMAA_PS_RUNNING	= 20H	: job is running
DRMAA_PS_SYSTEM_SUSPENDED	= 21H	: job is system suspended
DRMAA_PS_USER_SUSPENDED	= 22H	: job is user suspended
DRMAA_PS_DONE	= 30H	: job finished normally
DRMAA_PS_FAILED	= 40H	: job finished, but failed.

DRMAA should always get the status of job_id from DRM system,
unless the previous status has been DRMAA_PS_FAILED or DRMAA_PS_DONE and the status has been successfully cached. Terminated jobs get DRMAA_PS_FAILED status.

drmaa_ synchronize routine returns DRMAA_ERRNO_SUCCESS on success, otherwise
DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE,
DRMAA_ERRNO_AUTH_FAILURE, or
DRMAA_ERRNO_INVALID_JOB.

3.2.6 Auxiliary Routines

error_string drmaa_strerror (errno);

IN errno	/* Errno number (integer) */
RETURNS	/* Readable text version of errno (constant string) */

Get the error message text associated with the errno number.

contact drmaa_get_contact();

OUT contact	/* Current contact information for DRM system (string) */

drmaa_version(major, minor)
```
  OUT major                        /* major version number (non-negative integer) */
  OUT minor                        /* minor version number (non-negative integer) */
```

Returns the major and minor version numbers of the DRMAA library;
for DRMAA 1.0, 'major' is 1 and 'minor' is 0.

DRM_system drmaa_get_DRM_system( )
```
  RETURNS                          /* DRM system implementation information */
```

Output (string) is implementation dependent and could contain the DRM system and the implementation vendor as its parts.

3.3 List of DRMAA Errors

-------------- these are relevant to all sections ----------------

DRMAA_ERRNO_SUCCESS
Routine returned normally with success.

DRMAA_ERRNO_INTERNAL_ERROR
Unexpected or internal DRMAA error like memory allocation, system call failure, etc.

DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE
Could not contact DRM system for this request.

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

DRMAA_ERRNO_INVALID_ARGUMENT
The input value for an argument is invalid.

-------------- init and exit specific ---------------

DRMAA_ERRNO_INVALID_CONTACT_STRING
Initialization failed due to invalid contact string.

DRMAA_ERRNO_DEFAULT_CONTACT_STRING_ERROR
DRMAA could not use the default contact string to connect to DRM system.

DRMAA_ERRNO_DRMS_INIT_FAILED
Initialization failed due to failure to init DRM system.

DRMAA_ERRNO_ALREADY_ACTIVE_SESSION
Initialization failed due to existing DRMAA session.

DRMAA_ERRNO_NO_ACTIVE_SESSION
Exit routine failed because there is no active session.

DRMAA_ERRNO_DRMS_EXIT_ERROR
DRM system disengagement failed.

---------------- job attributes specific --------------
DRMAA_ERRNO_INVALID_ATTRIBUTE_FORMAT
The format for the job attribute value is invalid.

DRMAA_ERRNO_INVALID_ATTRIBUTE_VALUE
The value for the job attribute is invalid.

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

--------------------- job submission specific --------------

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

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

------------------------------ job control specific ----------------

DRMAA_ERRNO_INVALID_JOB
The job specified by the 'jobid' does not exist.

DRMAA_ERRNO_RESUME_INCONSISTENT_STATE
The job has not been suspended. The RESUME request will not be processed.

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

DRMAA_ERRNO_HOLD_INCONSISTENT_STATE
The job cannot be moved to a HOLD state.

DRMAA_ERRNO_RELEASE_INCONSISTENT_STATE
The job is not in a HOLD state.

DRMAA_ERRNO_EXIT_TIMEOUT
We have encountered a time-out condition for drmaa_synchronize or drmaa_wait.

DRMAA_ERRNO_NO_RUSAGE
This error code is returned by drmaa_wait() when a job has finished but no rusage and stat data could be provided.

4. Security Considerations

Security issues are not discussed in this document. The scheduling scenario described here assumes that security is handled at the point of job authorization/execution on a particular resource.

Author Information

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

Waiman Chan
waimanc@us.ibm.com
International Business Machines Corporation
2455 South Road
Poughkeepsie, NY 12601

Fritz Ferstl
fritz.ferstl@sun.com
Sun Microsystems GmbH
Dr.-Leo-Ritter-Str. 7
D-93049 Regensburg
Germany

Jeffrey T. Gardiner
gardiner@imaging.robarts.ca
Robarts Research Institute
PO Box 5015, 100 Perth Drive
London ON, N6A 5K8
Canada

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

Bill Nitzberg
bill@computer.org
Veridian, PBS Products
2672 Bayshore Pkwy, Suite 810
Mountain View, CA 94043

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

John Tollefsrud
j.t@sun.com
Sun Microsystems
200 Jefferson Drive UMPK29-302
Menlo Park, CA 94025