GWD-R
Distributed Resource Management
Application API (DRMAA) Working Group

Daniel Templeton, Sun Microsystems (maintainer)
Peter Tröger, Hasso-Plattner Institute
Roger Brobst, Cadence Design Systems
Andreas Haas*, Sun Microsystems
Hrabri Rajic*, Intel Americas Inc.
John Tollefsrud, Sun Microsystems
*co-chairs November, 2007

Distributed Resource Management Application API Java™ Language Bindings 1.0.1

Status of This Memo

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

Abstract

This document describes the Distributed Resource Management Application API (DRMAA) Java™ language bindings. The document is based on the implementation work of the DRMAA GWD-R document.

Contents

Tables

1. Introduction

This document describes the Java language binding for the DRMAA interface. This Java language binding was developed with the Java 2 Standard Edition™ 1.4.2 in mind, however it should be possible to implement with any Java platform version 1.2 or greater. This requirement stems from the use of the Collections API which was first introduced with Java Development Kit™ 1.2.

2. Notational 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 [BRADNER1]

3. Design Decisions

In order to make the Java language binding as familiar to programmers as possible, whenever possible, design elements were borrowed from common Java language APIs. The Java language binding makes use of an API/SPI factory pattern similar to the JAX Pack APIs. The Java language binding also abstracts exception handing to a single, declared, top-level exception as is done in the JDBC API. Properties in the Java language binding follow the standard JavaBean™ property pattern.

3.1 Service Provider Interface

The DRMAA Java language binding allows vendors to implement the DRM-specific binding classes required to interface with a given DRM without changing the outward facing API. By extending classes in the Java language binding and providing implementations of the various methods, a vendor can tailor his implementation to his needs. The vendor implementation SHOULD be completely transparent to the DRMAA application, however. The API hides the SPI and prevents the DRMAA application from needing to know anything about the underlying implementation.

4. Relationship to Other DRMAA Specifications

The Java language binding specification is related to both the Distributed Resource Management Application API Specification 1.0 (GFD.022) and the Distributed Resource Management Application API – IDL Binding 1.0 (DRMAA IDL). The former is the parent specification of all specifications in the DRMAA family and lays out the behavior and functionality that must be defined in this and all binding specifications. The later lays the ground work for the next generation of DRMAA specifications and builds on the former, providing specifics of syntax and semantics to be declared in this document. As the later specification evolves, this document will also evolve in order to remain in sync.

The DRMAA IDL specification suggests a mapping of abstract IDL constructs to elements of the particular language. For the Java Language Binding API, the following mappings apply:

IDL Construct	*Java Mapping*
long	int
long long	long
string	java.lang.String
boolean	boolean
const	public static final
module	Java package
interface	Java interface
exception	Java class derived from java.lang.Exception
raises	throws
valuetype	public class
factory	Java class constructor
JobControlAction enumeration	Session interface constants
JobProgramState enumeration	Session interface constants
JobSubmissionState enumeration	JobTemplate interface constants
StringList type	java.util.Set
OrderedStringList type	java.util.List
TimeAmount type	long
Dictionary type	java.util.Map
(readonly) IDL attribute	Java property with associated getter (and setter)

Table 1: IDL Mapping for the Java Language

5. The Java Language Binding API

The GFD.022 DRMAA Interface Specification was written originally with a slant towards a C binding. As such, several aspects of the DRMAA interface needed to be interpreted with liberty in order to better fit with an object-oriented language like the Java language. Among the aspects that changed are variable and method naming and the error structure. Details of this altering are described in the DRMAA IDL specification.

As suggested in the DRMAA IDL specification, some methods from the GFD.022 specification fail to appear in object-oriented bindings such as the Java language binding specification. The drmaa_get_attribute(), drmaa_set_attribute(), drmaa_get_vector_attribute(), drmaa_set_vector_attribute(), and drmaa_get_vector_attribute_names() methods are not needed because the Java language binding specification specifies a property *getter* and *setter* for each DRMAA attribute. A getter is a method for getting a property's value, and a setter is a method for setting a property's value. The advantage of this approach is that the property getters and setters allow for compile-time type checking of DRMAA attributes, and allow special

treatment of attributes which are better represented as something other than a String. Below is a table of the DRMAA IDL/Java property names and their corresponding Java language types.

DRMAA IDL/Java Property	*Java Language Type*
remoteCommand	java.lang.String
args	java.lang.String[]
jobSubmissionState	int
jobEnvironment	java.util.Map
workingDirectory	java.lang.String
jobCategory	java.lang.String
nativeSpecification	java.lang.String
email	java.lang.String[]
blockEmail	boolean
startTime	org.ggf.drmaa.PartialTimestamp
jobName	java.lang.String
inputPath	java.lang.String
outputPath	java.lang.String
errorPath	java.lang.String
joinFiles	boolean
transferFiles	org.ggf.drmaa.FileTransferMode
deadlineTime	org.ggf.drmaa.PartialTimestamp
hardWallclockTimeLimit	long
softWallclockTimeLimit	long
hardRunDurationLimit	long
softRunDurationLimit	long

Table 2: DRMAA Properties and Java Language Types

The setters and getters follow the JavaBean™ pattern for properties. For an attribute named *attribute* of type *Type*, the signature of the getter and setter would be:

```
public void setAttribute (Type value) throws DrmaaException;
public Type getAttribute ()
```

All property getters and setters MUST operate in a pass-by-value mode. For data types which are not natively pass-by-value, such as org.ggf.drmaa.FileTransferMode, the data MUST be copied so that the data structure stored by the Java language binding is decoupled from the data structure in the calling application.

Optional attributes are also represented by getters and setters. In conformance to the DRMAA IDL binding, the Java binding implementation MUST provide implementations in the SimpleJobTemplate class of getters and setters for all DRMAA attributes, both required and optional. The getter and setter implementations for optional attributes in the SimpleJobTemplate class MUST throw org.ggf.drmaa.UnsupportedAttributeException. The service provider implementation SHOULD then override the getters and setters for supported optional attributes with methods that operate normally.

The JobTemplate.**getAttributeNames()** method MUST return the names of all properties supported by the service provider implementation, including required, optional, and implementation specific attributes. In order for an application to get the values for all supported attributes of a JobTemplate instance, such as in a property sheet, the application should use introspection to call the appropriate getter for each attribute.

5.1 The Session Interface

The main class in the Java language binding is the Session interface. It represents the majority of the functionality defined by the DRMAA Interface Specification. Please consult GFD.022 section 3.1.2 for further details about the DRMAA session concept. The Session interface has the following structure:

```
public abstract interface org.ggf.drmaa.Session {
    public static final int SUSPEND = 0;
    public static final int RESUME = 1;
    public static final int HOLD = 2;
    public static final int RELEASE = 3;
    public static final int TERMINATE = 4;
    public static final java.lang.String JOB_IDS_SESSION_ALL =
       "DRMAA_JOB_IDS_SESSION_ALL";
    public static final java.lang.String JOB_IDS_SESSION_ANY =
       "DRMAA_JOB_IDS_SESSION_ANY";
    public static final long TIMEOUT_WAIT_FOREVER = -1L;
    public static final long TIMEOUT_NO_WAIT = 0L;
    public static final int UNDETERMINED = 0x00;
    public static final int QUEUED_ACTIVE = 0x10;
    public static final int SYSTEM_ON_HOLD = 0x11;
    public static final int USER_ON_HOLD = 0x12;
    public static final int USER_SYSTEM_ON_HOLD = 0x13;
    public static final int RUNNING = 0x20;
    public static final int SYSTEM_SUSPENDED = 0x21;
    public static final int USER_SUSPENDED = 0x22;
    public static final int USER_SYSTEM_SUSPENDED = 0x23;
    public static final int DONE = 0x30;
    public static final int FAILED = 0x40;
    public void init(java.lang.String contactString)
       throws org.ggf.drmaa.DrmaaException;
    public void exit() throws org.ggf.drmaa.DrmaaException;
    public org.ggf.drmaa.JobTemplate createJobTemplate()
       throws org.ggf.drmaa.DrmaaException;
    public void
```

```
        deleteJobTemplate(org.ggf.drmaa.JobTemplate jobTemplate)
        throws org.ggf.drmaa.DrmaaException;
    public java.lang.String
        runJob(org.ggf.drmaa.JobTemplate jobTemplate)
        throws org.ggf.drmaa.DrmaaException;
    public java.util.List
        runBulkJobs(org.ggf.drmaa.JobTemplate jobTemplate,
                    int beginIndex, int endIndex, int step)
        throws org.ggf.drmaa.DrmaaException;
    public void control(java.lang.String jobName, int operation)
        throws org.ggf.drmaa.DrmaaException;
    public void synchronize(java.util.List jobList,
                            long timeout, boolean dispose)
        throws org.ggf.drmaa.DrmaaException;
    public org.ggf.drmaa.JobInfo
        wait(java.lang.String jobName, long timeout)
        throws org.ggf.drmaa.DrmaaException;
    public int getJobProgramStatus(java.lang.String jobName)
        throws org.ggf.drmaa.DrmaaException;
    public java.lang.String getContact()
        throws org.ggf.drmaa.DrmaaException;
    public org.ggf.drmaa.Version getVersion()
        throws org.ggf.drmaa.DrmaaException;
    public java.lang.String getDrmsInfo()
        throws org.ggf.drmaa.DrmaaException;
    public java.lang.String getDrmaaImplementation()
        throws org.ggf.drmaa.DrmaaException;
}
```

All methods of the Session interface MAY raise the following exceptions in addition to any explicitly listed in the method descriptions below:

- java.lang.OutOfMemoryError – as described in the Java Language Specification. This exception replaces the OutOfMemoryException from the DRMAA IDL specification.
- DrmCommunicationException – The DRMS could not be contacted.
- AuthorizationException – the session owner does not have permission to perform the chosen operation.
- Java.lang.IllegalArgumentException – as described in the Java Language Specification. This exception replaces the InvalidArgumentException from the DRMAA IDL specification. It MAY only be raised by methods with input arguments.
- InternalException – due to an error in the DRMAA implementation, the chosen operation could not be performed.

5.1.1 SUSPEND

The **SUSPEND** constant is used by the **control()** method to indicate that the given job should be suspended.

5.1.2 RESUME

The **RESUME** constant is used by the **control()** method to indicate that the given job should be resumed.

5.1.3 HOLD

The **HOLD** constant is used by the **control()** method to indicate that the given job should be placed into a hold state.

5.1.4 RELEASE

The **RELEASE** constant is used by the **control()** method to indicate that the given job should be released from its hold state.

5.1.5 TERMINATE

The **TERMINATE** constant is used by the **control()** method to indicate that the given job should be terminated.

5.1.6 JOB_IDS_SESSION_ALL

The **JOB_IDS_SESSION_ALL** constant is used by the **control()** and **synchronize()** methods to indicate that a method call should operate on all jobs in the session at submission time, minus any jobs that go out of scope during the run time of the operation. For example: If a job was in the session at the time of calling **synchronize(JOB_IDS_SESSION_ALL)**, and it gets reaped during the operation, the **synchronize()** call will not fail. The usage of control() or synchronize() with JOB_IDS_SESSION_ALL on an empty session SHALL result in a successful call result without any further effect. In the case that a call with *JOB_IDS_SESSION_ALL* fails for a partial set of the jobs in the session, the implementation SHALL throw an *InternalException*. The error text of the exception should explain the problem in detail and may give an idea of the current status of the session.

5.1.7 JOB_IDS_SESSION_ANY

The **JOB_IDS_SESSION_ANY** constant is used by the **wait()** method to indicate that a method call may operate on any job currently in the **RUNNING** state in the session.

5.1.8 TIMEOUT_WAIT_FOREVER

The **TIMEOUT_WAIT_FOREVER** constant is used by the **wait()** and **synchronize()** methods to indicate that a method call should not return until the given job or jobs have entered the **DONE** or **FAILED** state.

5.1.9 TIMEOUT_NO_WAIT

The **TIMEOUT_NO_WAIT** constant is used by the **wait()** and **synchronize()** methods to indicate that a method call should return immediately if the given job or jobs have not yet entered the **DONE** or **FAILED** state.

5.1.10 UNDETERMINED

The **UNDETERMINED** constant is used by the **getJobProgramStatus()** method to indicate that the job's current state cannot be determined.

5.1.11 QUEUED_ACTIVE

The **QUEUED_ACTIVE** constant is used by the **getJobProgramStatus()** method to indicate that the job is queued and waiting to be scheduled.

5.1.12 SYSTEM_ON_HOLD

The **SYSTEM_ON_HOLD** constant is used by the **getJobProgramStatus()** method to indicate that the job has been placed on hold by the system or administrator.

5.1.13 USER_ON_HOLD

The **USER_ON_HOLD** constant is used by the **getJobProgramStatus()** method to indicate that the job has been placed on hold by a user.

5.1.14 USER_SYSTEM_ON_HOLD

The **USER_SYSTEM_ON_HOLD** constant is used by the **getJobProgramStatus()** method to indicate that the job has been placed on hold by both the system or administrator and a user.

5.1.15 RUNNING

The **RUNNING** constant is used by the **getJobProgramStatus()** method to indicate that the job has been scheduled and is running.

5.1.16 SYSTEM_SUSPENDED

The **SYSTEM_SUSPENDED** constant is used by the **getJobProgramStatus()** method to indicate that the job has been suspended by the system or administrator.

5.1.17 USER_SUSPENDED

The **USER_SUSPENDED** constant is used by the **getJobProgramStatus()** method to indicate that the job has been suspended by a user.

5.1.18 USER_SYSTEM_SUSPENDED

The **USER_SYSTEM_SUSPENDED** constant is used by the **getJobProgramStatus()** method to indicate that the job has been suspended by both the system or administrator and a user.

5.1.19 DONE

The **DONE** constant is used by the **getJobProgramStatus()** method to indicate that the job has finished normally.

5.1.20 FAILED

The **FAILED** constant is used by the **getJobProgramStatus()** method to indicate that the job exited abnormally before finishing.

5.1.21 init

The **init()** method MUST do whatever work is required to initialize a DRMAA session for use. The *contactString* parameter is an implementation-dependent string that may be used to specify which DRM system to use. This method must be called before any other DRMAA calls, except for the **getDrmsInfo()**, **getDrmaaImplementation()**, and **getContact()** methods of the Session interface**.**

If contact is `null` or empty, the default DRM system SHOULD be used, provided there is only one DRMS available. If contact is `null` or empty, and more than one DRMS is available, **init()** SHALL throw a `NoDefaultContactStringSelectedException`. **init()** SHOULD be called only once, by only one of the threads. The main thread is recommended. A call to **init()** by another thread or additional calls to **init()** by the same thread SHOULD throw an `AlreadyActiveSessionException`.

In the case that a DRMAA library implementation needs to perform non-thread-safe operations, it SHOULD perform them in the implementation of the init() operation, in order to ensure thread-safe operations for all other DRMAA methods.

5.1.21.1 Parameters

`contactString` - implementation-dependent string that may be used to specify which DRM system to use. If `null` or empty, the DRMAA implementation will select the default DRM if there is only one DRMS available.

5.1.21.2 Throws

`DrmaaException` - MAY be be one of the following:

- DrmsInitException – failed while initializing the session.
- InvalidContactStringException – the *contact* parameter is invalid.
- AlreadyActiveSessionException – the session has already been initialized.
- DefaultContactStringException – the *contact* parameter is `null` or empty and the default contact string could not be used to connect to the DRMS.
- NoDefaultContactStringSelectedException – the *contact* parameter is `null` or empty and more than one DRMS is available.

5.1.22 exit

The **exit()** method MUST do whatever work is required to disengage from the DRM system and allow the DRMAA implementation to perform any necessary internal cleanup. This routine ends the current DRMAA session but SHALL NOT affect any jobs (e.g., queued and running jobs

remain queued and running). Any JobTemplate instances which have not yet been deleted become invalid after **exit()** is called, even after a subsequent call to **init()**. **exit()** SHOULD be called only once, by only one of the threads. Additional calls to **exit()** beyond the first SHALL throw a NoActiveSessionException, until a subsequent call to **init()**.

5.1.22.1 Throws

`DrmaaException` - MAY be one of the following:

- DrmsExitException – failed while exiting the session.
- NoActiveSessionException – the session has not been initialized or **exit()** has already been called

5.1.23 createJobTemplate

The **createJobTemplate()** method SHALL return a new JobTemplate instance. The job template is used to set the defining characteristics for jobs to be submitted. Once the job template has been created, it SHOULD also be deleted (via **deleteJobTemplate()**) when no longer needed. Failure to do so may result in a memory leak.

5.1.23.1 Returns

The **createJobTemplate()** method SHALL return a blank JobTemplate object. In service provider DRMAA implementations for Java SE 5.0 or later, the concrete service provider Session implementation SHOULD override the return type of this method to be that of the service provider job template implementation if one exists.

5.1.23.2 Throws

`DrmaaException` - MAY be one of the following:

- NoActiveSessionException – the session has not been initialized

5.1.24 deleteJobTemplate

The **deleteJobTemplate()** method is used to deallocate a job template, and SHALL perform all necessary steps required to free all memory associated with the given job template instance. If a DRMAA implementation provides a finalizer method for the JobTemplate implementation, the implementation of this method MAY be empty.

This method SHALL have no effect on running jobs. This method MUST only work on JobTemplate instances that were created with the **createJobTemplate()** method and have not previously been deleted with the **deleteJobTemplate()** method and MUST otherwise throw an InvalidJobTemplateException.

5.1.24.1 Parameters

`jt` - the JobTemplate to delete.

5.1.24.2 Throws

`DrmaaException` - MAY be one of the following:

- NoActiveSessionException – the session has not been initialized
- InvalidJobTemplateException – the given job template was not created with **createJobTemplate()** or has already been deleted.

5.1.25 runJob

The **runJob()** method SHALL submit a job with attributes defined in the job template given as a parameter. The returned job identifier SHOULD be a String identical to that returned from the underlying DRM system. This method MUST only work on JobTemplate instances that were created with the **createJobTemplate()** method and have not previously been deleted with the **deleteJobTemplate()** method and MUST otherwise throw an InvalidJobTemplateException.

5.1.25.1 Parameters

`jt` - the job template to be used to create the job.

5.1.25.2 Returns

The **runJob()** method SHOULD return a job identifier string identical to that returned from the underlying DRM system.

5.1.25.3 Throws

`DrmaaException` - MAY be one of the following:

- TryLaterException – the request could not be processed due to excessive system load.
- DeniedByDrmException – the DRMS rejected the job. The job will never be accepted due to job template or DRMS configuration settings.
- InvalidJobTemplateException – the given job template was not created with **createJobTemplate()** or has already been deleted.
- NoActiveSessionException – the session has not been initialized or **exit()** has already been called.

5.1.26 runBulkJobs

The **runBulkJobs()** method SHALL submit a set of parametric jobs, dependent on the implied loop index, each with attributes defined in the given job template. Each job in the set is identical except for its index. The first parametric job has an index equal to *beginIndex*. The next job has an index equal to *beginIndex* + *step*, and so on. The last job has an index equal to *beginIndex* + **n** * *step*, where **n** is equal to (*endIndex* – *beginIndex*) / *step*. Note that the value of the last job's index may not be equal to *endIndex* if the difference between *beginIndex* and *endIndex* is not evenly divisible by *step*. The smallest valid value for *beginIndex* is 1. The largest valid value for *endIndex* is language dependent. The *beginIndex* value must be less than or equal to the *endIndex* value, and only positive index numbers are allowed. The index number can be determined by the job in an implementation-specific fashion. The returned job identifiers SHOULD be Strings identical to those returned from the underlying DRM system.

The JobTemplate interface defines a **PARAMETRIC_INDEX** placeholder for use in specifying paths. This placeholder is used to represent the individual identifiers of the tasks submitted through this method.

This method MUST only work on JobTemplate instances that were created by the **createJobTemplate()** method and have not previously been deleted by the **deleteJobTemplate()** or **exit()** method and MUST otherwise throw an InvalidJobTemplateException.

5.1.26.1 Parameters

`jobTemplate` - the job template to be used to create the job.

`beginIndex` - the starting value for the loop index.

`EndIndex` - the terminating value for the loop index.

`step` - the value by which to increment the loop index each iteration.

5.1.26.2 Returns

The **runBulkJobs()** method SHOULD return a list of job identifier Strings identical to that returned by the underlying DRM system.

5.1.26.3 Throws

`DrmaaException` - MAY be one of the following:

- TryLaterException – the request could not be processed due to excessive system load.
- DeniedByDrmException – the DRMS rejected the job. The job will never be accepted due to job template or DRMS configuration settings.
- InvalidJobTemplateException – the given job template was not created with **createJobTemplate()** or has already been deleted.
- NoActiveSessionException – the session has not been initialized or **exit()** has already been called.

5.1.27 control

The **control()** method SHALL hold, release, suspend, resume, or kill the job identified by *jobName*. If *jobName* is **JOB_IDS_SESSION_ALL**, then this method SHALL act on all jobs submitted during this DRMAA session up to the moment **control()** is called. See 5.1.6.

To avoid thread races in multi-threaded 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 legal values for *operation* and their meanings SHALL be:

- **SUSPEND**: stop the job,
- **RESUME**: (re)start the job,
- **HOLD**: put the job on-hold,

- **RELEASE**: release the hold on the job, and
- **TERMINATE**: kill the job.

This method SHALL return once the action has been acknowledged by the DRM system, but MAY return before the action has been completed.

Some DRMAA implementations MAY allow this method to be used to control jobs submitted externally to the DRMAA session, such as jobs submitted by other DRMAA session in other DRMAA implementations or jobs submitted via native utilities.

5.1.27.1 Parameters

`jobName` - The id of the job to control.

`operation` - the control action to be taken.

5.1.27.2 Throws

`DrmaaException` - MAY be one of the following:

- ResumeInconsistentStateException – the job is not in a state from which is can be resumed.
- SuspendInconsistentStateException – the job is not in a state from which is can be suspended.
- HoldInconsistentStateException – the job is not in a state from which is can be held.
- ReleaseInconsistentStateException – the job is not in a state from which is can be released.
- InvalidJobException – the job id does not represent a valid job.
- NoActiveSessionException – the session has not been initialized or **exit()** has already been called.

5.1.28 synchronize

This method SHALL wait until all jobs specified by *jobList* have finished execution. If *jobList* contains **JOB_IDS_SESSION_ALL**, then this method SHALL wait for all jobs submitted during this DRMAA session up to the moment **synchronize()** is called. See 5.1.6.

In the case where a call with **JOB_IDS_SESSION_ALL** fails for a partial set of the jobs in the session, the implementation SHALL throw an InternalException. The error text of the exception should explain the problem in detail and may give an idea of the current status of the session.

To avoid thread race conditions in multi-threaded 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.

To prevent blocking indefinitely in this call, the caller may use a timeout specifying how many seconds to block in this call. The value **TIMEOUT_WAIT_FOREVER** may be specified to wait indefinitely for a result. The value **TIMEOUT_NO_WAIT** may be specified to return immediately. If the call exits before the timeout has elapsed, all the jobs have been waited on or there was an

interrupt. If the invocation exits on timeout, an ExitTimeoutException SHALL be thrown. The caller should check system time before and after this call in order to be sure of how much time has passed.

If at any time during the call to **synchronize()**, no jobs are active in the session, the call to **synchronize()** will return immediately.

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 set to `true`, the DRM SHALL dispose of the job's data record. If set to `false`, the data record SHALL be left for future access via the **wait()** method. Because a DRMAA implementation is not required to retain information about jobs which have bean reaped, the routine is not required to, but MAY distinguish between non-existent and reaped jobs. If the routine successfully validates a jobId for an already reaped job, it MAY return successfully.

5.1.28.1 Parameters

`jobList` - the ids of the jobs to synchronize.

`timeout` - the maximum number of seconds to wait.

`dispose` - specifies how to treat reaping information.

5.1.28.2 Throws

`DrmaaException` - MAY be one of the following:

- ExitTimeoutException – the call was interrupted before all given jobs finished.
- InvalidJobException – the job id does not represent a valid job.
- NoActiveSessionException – the session has not been initialized or **exit()** has already been called.

5.1.29 wait

This method SHALL wait for a job with *jobName* to finish execution or fail. If **JOB_IDS_SESSION_ANY** is provided as the *jobName*, this method SHALL wait for any job submitted during this DRMAA session up to the moment **wait()** is called. At any time during a call to **wait()** with **JOB_IDS_SESSION_ANY** as the *jobName* parameter, if no jobs are active in the session, the call to **wait()** SHALL fail, throwing an InvalidJobException. This method is modeled on the wait3 POSIX routine. Only one invocation of the **wait()** method for a given job id MAY succeed. The others MUST throw an InvalidJobException.

The *timeout* value SHALL be used to specify the desired behavior when a result is not immediately available. The value, **TIMEOUT_WAIT_FOREVER**, may be specified to wait indefinitely for a result. The value, **TIMEOUT_NO_WAIT**, may be specified to return immediately. 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* seconds, either the job has been waited on successfully or there was an interrupt. If the invocation exits on timeout, an ExitTimeoutException SHALL be thrown.

The caller should check system time before and after this call in order to be sure how much time has passed.

This method SHALL reap job data records on a successful call, so any subsequent calls to **wait()** will fail, throwing an InvalidJobException, meaning that the job's data record has already been reaped. This exception is the same as if the job were unknown. (The only case where **wait()** can be successfully called on a single job more than once is when the previous call to **wait()** timed out before the job finished.)

In a multi-threaded environment with a **wait()** call using **JOB_IDS_SESSION_ANY**, only the active thread gets the status of the finished or failed job, while the other threads continue waiting. If there are no more running or completed jobs left in the session, all remaining waiting threads SHOULD fail with an InvalidJobException. If thread A is waiting for a specific job, and another thread, thread B, waiting for that same job or with *JOB_IDS_SESSION_ANY,* receives notification that the job has finished, thread A SHOULD fail with an *InvalidJobException*. At any time during a call to *wait()* with *JOB_IDS_SESSION_ANY* as the *jobName* parameter, if no jobs are active in the session, the call to *wait()* SHALL fail, throwing an *InvalidJobException*.

When successful, the resource usage information for the job SHALL be provided as a java.util.Map of usage parameter names and their values in the returned JobInfo instance. The values contain the amount of resources consumed by the job and are implementation defined.

5.1.29.1 Parameters

`jobName` - the id of the job for which to wait.

`timeout` - the maximum number of seconds to wait.

5.1.29.2 Returns

This method SHALL return the resource usage and status information as a JobInfo instance.

5.1.29.3 Throws

`DrmaaException` - MAY be one of the following:

- ExitTimeoutException – the call was interrupted before the given job finished.
- InvalidJobException – the job id does not represent a valid job.
- NoActiveSessionException – the session has not been initialized or **exit()** has already been called.

5.1.30 getJobStatus

The **getJobStatus()** method SHALL return the program status of the job identified by *jobName*. The possible values returned from this method are:

- **UNDETERMINED**: process status cannot be determined,
- **QUEUED_ACTIVE**: job is queued and waiting to be scheduled,
- **SYSTEM_ON_HOLD**: has been placed on hold by the system or the administrator,
- **USER_ON_HOLD**: job has been placed on hold by a user,

- **USER_SYSTEM_ON_HOLD**: job has been placed on hold by both the system or administrator and a user,
- **RUNNING**: job has been scheduled and is running,
- **SYSTEM_SUSPENDED**: job has been suspended by the system or administrator,
- **USER_SUSPENDED**: job has been suspended by a user,
- **USER_SYSTEM_SUSPENDED**: job has been suspended by both the system or administrator and a user,
- **DONE**: job finished normally, and
- **FAILED**: job exited abnormally before finishing.

The DRMAA implementation MUST always get the status of the job from the DRM system unless the status has already been determined to be **FAILED** or **DONE** and the status has been successfully cached. Terminated jobs return a **FAILED** status. It is up to the implementation to determine whether this method is capable of operating on jobs submitted outside of the current DRMAA session.

A DRMAA implementation is not required to be able to return all of the job state values in the above list. If a given job state has no representation in the underlying DRMS, the DRMAA implementation MAY ignore that job state value. All DRMAA implementations MUST, however, define **all** listed job state constants, including those for unused job states. An implementation SHOULD NOT return any job state value other than those listed above.

5.1.30.1 Parameters

`jobName` - the id of the job whose status is to be retrieved.

5.1.30.2 Returns

The **getStatus()** SHALL return the program status.

5.1.30.3 Throws

`DrmaaException` - MAY be one of the following:

- InvalidJobException – the job id does not represent a valid job.
- NoActiveSessionException – the session has not been initialized or **exit()** has already been called.

5.1.31 getJobProgramStatus

This method is deprecated and has been replaced by the **getJobStatus()** method. For reasons of backwards compatibility with applications written against the 1.0 version of the specification, this method SHALL be supported and marked as deprecated, and it SHALL be a simple wrapper around the **getJobStatus()** method.

5.1.31.1 Parameters

`jobName` - the id of the job whose status is to be retrieved.

5.1.31.2 Returns

The **getStatus()** SHALL return the program status.

5.1.31.3 Throws

`DrmaaException` - MAY be one of the following:

- InvalidJobException – the job id does not represent a valid job.
- NoActiveSessionException – the session has not been initialized or **exit()** has already been called.

5.1.32 getContact

If called before **init()**, this method SHALL return a comma delimited String containing the default DRMAA implementation contact Strings, one element per DRM system available. A contact string represents a specific installation of a specific DRM system, e.g. a Condor central manager machine at a given IP address or a Sun Grid Engine 'root' and 'cell'.

If called after **init()**, this method SHALL return the contact String for the DRM system to which the session is attached.

The returned Strings are always implementation dependent and SHOULD NOT be interpreted by the application.

5.1.32.1 Returns

This method SHALL return the current contact information for the DRM system or a comma delimited list of possible contact Strings.

5.1.33 getVersion

This method SHALL return a Version instance containing the major and minor version numbers of the DRMAA library. For DRMAA 1.0, major is 1 and minor is 0. This method may not be called before **init()** has been called.

5.1.33.1 Returns

This method SHALL return the version number as a Version object.

5.1.34 getDrmsInfo

If called before **init()**, this method SHALL return a comma delimited list of DRM system identifiers, one element per DRM system implementation provided. A DRM system identifier denotes a specific type of DRM system, e.g. Sun Grid Engine.

If called after **init()**, this method SHALL return the selected DRM system identifier. The returned Strings are implementation dependent and SHOULD NOT be interpreted by the application.

5.1.34.1 Returns

This method SHALL return the DRM system identifier information.

5.1.35 getDrmaaImplementation

If called before **init()**, this method SHALL return a comma delimited list of DRMAA implementations, one element for each DRMAA implementation provided. A DRMAA implementation string denotes a specific version of a DRM system, e.g. Grid Engine 6.0u9.

If called after **init()**, this method SHALL return the selected DRMAA implementation. The returned Strings are implementation dependent. They MAY contain the DRM system identifier as a component and SHOULD NOT be interpreted by the application.

5.1.35.1 Returns

This method SHALL return the DRMAA implementation information.

5.2 The SessionFactory Class

In order to enable a Java language binding implementation to be supported by multiple different vendors, a factory class is needed to allow a DRMAA application to retrieve a vendor specific implementation of the Session interface. The SessionFactory class serves this purpose and additionally allows the vendor the freedom to return different Session implementations depending on the need. The structure of the SessionFactory class is as follows:

```
public abstract class org.ggf.drmaa.SessionFactory {
    public static org.ggf.drmaa.SessionFactory getFactory()
    public abstract org.ggf.drmaa.Session getSession()
}
```

It is likely that with a future version of this specification, the SessionFactory class will include a method to explicitly request a specific service provider implementation.

5.2.1 getFactory

This method MUST return a SessionFactory instance appropriate for the DRM in use. This method MUST use the “org.ggf.drmaa.SessionFactory” property to find the appropriate SessionFactory implemention class. It MUST first look in the system properties. If the property is not present, the method MUST look in $java.home/lib/drmaa.properties. If the property still isn't found, the method MUST search the classpath for a META-INF/services/org.ggf.drmaa.SessionFactory resource. If the property still has not been found, the method MUST throw an Error.

5.2.1.1 Returns

This method MUST return a SessionFactory instance appropriate for the DRM in use.

5.2.1.2 Throws

`Error` – No SessionFactory implementation class could be found.

5.2.2 getSession

This method MUST return a Session instance appropriate for the DRM in use.

5.2.2.1 Returns

This method MUST return a Session instance appropriate for the DRM in use. In service provider DRMAA implementations for Java SE 5.0 or later, the concrete service provider SessionFactory implementation SHOULD override the return type of this method to be that of the concrete service provider Session implementation if one exists.

5.3 The JobTemplate Interface

In order to define the attributes associated with a job, a DRMAA application uses the JobTemplate interface. Instances of such templates are created via the active Session implementation. A DRMAA application gets a JobTemplate instance from the active Session instance, specifies in the template any required job parameters, and the passes the template back to the Session instance when requesting that a job be executed. When finished, the DRMAA application SHOULD call the Session.**deleteJobTemplate()** method to allow the underlying implementation to free any resources bound to the JobTemplate instance. Please refer also to GFD.022 section 3.1.4 to 3.1.6 for more information regarding precedence rules, site-specific requirements and job evaluation.

The structure of the JobTemplate interface is as follows:

```
public abstract interface org.ggf.drmaa.JobTemplate {
    public static final int HOLD_STATE = 0;
    public static final int ACTIVE_STATE = 1;
    public static final java.lang.String HOME_DIRECTORY =
       "$drmaa_hd_ph$";
    public static final java.lang.String WORKING_DIRECTORY =
       "$drmaa_wd_ph$";
    public static final java.lang.String PARAMETRIC_INDEX =
       "$drmaa_incr_ph$";
    public void setRemoteCommand(java.lang.String command)
       throws org.ggf.drmaa.DrmaaException;
    public java.lang.String getRemoteCommand()
       throws org.ggf.drmaa.DrmaaException;
    public void setArgs(java.util.List args)
       throws org.ggf.drmaa.DrmaaException;
    public java.util.List getArgs()
       throws org.ggf.drmaa.DrmaaException;
    public void setJobSubmissionState(int state)
       throws org.ggf.drmaa.DrmaaException;
    public int getJobSubmissionState()
       throws org.ggf.drmaa.DrmaaException;
    public void setJobEnvironment(java.util.Map env)
       throws org.ggf.drmaa.DrmaaException;
```

```
public java.util.Map getJobEnvironment()
   throws org.ggf.drmaa.DrmaaException;
public void setWorkingDirectory(java.lang.String wd)
   throws org.ggf.drmaa.DrmaaException;
public java.lang.String getWorkingDirectory()
   throws org.ggf.drmaa.DrmaaException;
public void setJobCategory(java.lang.String category)
   throws org.ggf.drmaa.DrmaaException;
public java.lang.String getJobCategory()
   throws org.ggf.drmaa.DrmaaException;
public void setNativeSpecification(java.lang.String spec)
   throws org.ggf.drmaa.DrmaaException;
public java.lang.String getNativeSpecification()
   throws org.ggf.drmaa.DrmaaException;
public void setEmail(java.util.Set email)
   throws org.ggf.drmaa.DrmaaException;
public java.util.Set getEmail()
   throws org.ggf.drmaa.DrmaaException;
public void setBlockEmail(boolean blockEmail)
   throws org.ggf.drmaa.DrmaaException;
public boolean getBlockEmail()
   throws org.ggf.drmaa.DrmaaException;
public void setStartTime(org.ggf.drmaa.PartialTimestamp startTime)
   throws org.ggf.drmaa.DrmaaException;
public org.ggf.drmaa.PartialTimestamp getStartTime()
   throws org.ggf.drmaa.DrmaaException;
public void setJobName(java.lang.String name)
   throws org.ggf.drmaa.DrmaaException;
public java.lang.String getJobName()
   throws org.ggf.drmaa.DrmaaException;
public void setInputPath(java.lang.String inputPath)
   throws org.ggf.drmaa.DrmaaException;
public java.lang.String getInputPath()
   throws org.ggf.drmaa.DrmaaException;
public void setOutputPath(java.lang.String outputPath)
   throws org.ggf.drmaa.DrmaaException;
public java.lang.String getOutputPath()
   throws org.ggf.drmaa.DrmaaException;
public void setErrorPath(java.lang.String errorPath)
   throws org.ggf.drmaa.DrmaaException;
public java.lang.String getErrorPath()
   throws org.ggf.drmaa.DrmaaException;
public void setJoinFiles(boolean joinFiles)
   throws org.ggf.drmaa.DrmaaException;
public boolean getJoinFiles()
   throws org.ggf.drmaa.DrmaaException;
public void setTransferFiles(org.ggf.drmaa.FileTransferMode mode)
   throws org.ggf.drmaa.DrmaaException;
public org.ggf.drmaa.FileTransferMode getTransferFiles()
   throws org.ggf.drmaa.DrmaaException;
```

```
    public void setDeadlineTime(org.ggf.drmaa.PartialTimestamp
       deadline) throws org.ggf.drmaa.DrmaaException;
    public org.ggf.drmaa.PartialTimestamp getDeadlineTime()
       throws org.ggf.drmaa.DrmaaException;
    public void setHardWallclockTimeLimit(long limit)
       throws org.ggf.drmaa.DrmaaException;
    public long getHardWallclockTimeLimit()
       throws org.ggf.drmaa.DrmaaException;
    public void setSoftWallClockTimeLimit(long limit)
       throws org.ggf.drmaa.DrmaaException;
    public long getSoftWallClockTimeLimit()
       throws org.ggf.drmaa.DrmaaException;
    public void setHardRunDurationLimit(long limit)
       throws org.ggf.drmaa.DrmaaException;
    public long getHardRunDurationLimit()
       throws org.ggf.drmaa.DrmaaException;
    public void setSoftRunDurationLimit(long limit)
       throws org.ggf.drmaa.DrmaaException;
    public long getSoftRunDurationLimit()
       throws org.ggf.drmaa.DrmaaException;
    public java.util.Set getAttributeNames()
       throws org.ggf.drmaa.DrmaaException;
}
```

All methods of the JobTemplate interface MAY raise the following exceptions in addition to any explicitly listed in the method descriptions below:

- NoActiveSessionException – the session has not been initialized or **exit()** has already been called.
- java.lang.OutOfMemoryError – as described in the Java Language Specification. This exception replaces the OutOfMemoryException from the DRMAA IDL specification.
- DrmCommunicationException – The DRMS could not be contacted.
- AuthorizationException – the session owner does not have permission to perform the chosen operation.
- Java.lang.IllegalArgumentException – as described in the Java Language Specification. This exception replaces the InvalidArgumentException from the DRMAA IDL specification. It MAY only be raised by methods with input arguments.
- InternalException – due to an error in the DRMAA implementation, the chosen operation could not be performed.

The JobTemplate implementation MUST support two types of exceptions for the setter operations:

- InvalidAttributeValueException – The value is invalid for the job template property, e.g. a start time that is in the past.
- ConflictingAttributeValuesException – the attribute value conflicts with a previously set attribute value.

In most cases, a DRMAA implementation will require that job templates be created through the Session.**createJobTemplate()** method. In those cases, passing a template created other than via this method to the Session.**deleteJobTemplate()**, Session.**runJob()**, or Session.**runBulkJobs()** methods MUST result in an InvalidJobTemplateException being thrown.

A JobTemplate instance MUST override the **toString()** method. The String returned from this method MUST contain the values of all set properties.

Access to attribute values MUST operate in a pass-by-value mode. Setters for non-primitive, mutable properties SHOULD therefore store a copy of the new value instead of storing the original object.

In the DRMAA job template there is a distinction between mandatory and optional attributes. A Java language binding implementation MUST provide implementations for all DRMAA attributes, both required and optional. The setter and getter implementations for optional attributes MUST throw UnsupportedAttributeException. The service provider implementation SHOULD then override the setters and getters for supported optional attributes with methods that operate normally.

A service provider DRMAA implementation is allowed to add implementation-specific attributes. The **getAttributeNames()** method SHALL return the names of all job template attributes supported by the service provider implementation, including required, optional, and implementation-specific attributes. To access implementation-specific job template attributes, one should cast the JobTemplate to a more specific JobTemplate subtype. In order to get the values for all supported attributes, such as in a property sheet, one should use introspection to call the appropriate setter and getter for each attribute.

5.3.1 HOLD_STATE

The **HOLD_STATE** constant represents a value for the jobSubmissionState property which means the job may be queued, but it is not eligible to run.

5.3.2 ACTIVE_STATE

The **ACTIVE_STATE** constant represents a value for the jobSubmissionState property which means the job is eligible to run.

5.3.3 HOME_DIRECTORY

The **HOME_DIRECTORY** constant is a place holder used to represent the user's home directory when building paths for the workingDirectory, inputPath, outputPath, and errorPath properties.

5.3.4 WORKING_DIRECTORY

The **WORKING_DIRECTORY** constant is a place holder used to represent the current working directory when building paths for the inputPath, outputPath, and errorPath properties.

5.3.5 PARAMETRIC_INDEX

The **PARAMETRIC_INDEX** constant is a place holder used to represent the id of the current parametric job subtask when building paths for the workingDirectory, inputPath, outputPath, and errorPath properties.

5.3.6 getAttributeNames

This method SHALL return the list of supported property names. This list includes supported DRMAA reserved property names (both required and optional) and implementation-specific property names.

5.3.6.1 Returns

This method SHALL return the list of supported property names.

5.3.6.2 Throws

`DrmaaException` - MAY be one of the following:

- NoActiveSessionException – the session has not been initialized or **exit()** has already been called

5.3.7 Getters

For each property listed in Table 2: DRMAA Properties and Java Language Types, the JobTemplate interface has a corresponding getter. Each getter is of the form "public <propertyType> get<propertyName>()."

5.3.7.1 Returns

The getter methods each return the current value of the corresponding property in the job template. All non-primitive, mutable return values MUST be copies of the originals.

5.3.8 Setters

For each property listed in Table 2: DRMAA Properties and Java Language Types, the JobTemplate class has a corresponding setter. Each setter is of the form "public void set<propertyName>(<propertyType> value)." Setters for non-primitve, mutable properties MUST store a copy of the new value rather than storing the original object.

5.3.8.1 Parameters

value – the value to which the property should be set in the job template.

5.3.8.2 Throws

`DrmaaException` - MAY be one of the following:

- InvalidAttributeValueException – the property value is invalid for the property, e.g. a startTime that is in the past.

- ConflictingAttributeValuesException – the property value conflicts with a previously set property value.

5.3.9 Required Properties

5.3.9.1 remoteCommand

The command that should be executed on the remote host. In case this parameter contains path information, it MUST be seen as relative to the execution host file system and is therefore evaluated there. The property value SHOULD NOT relate to binary file management or file staging activities.

5.3.9.2 args

The list of command-line arguments for the job to be executed.

5.3.9.3 jobSubmissionState

Defines the state of the job at submission time. The value may either be **HOLD_STATE** or **ACTIVE_STATE**.

5.3.9.4 jobEnvironment

The environment values that define the remote environment. The values MUST override the remote environment values if there is a collision. If this is not possible, the behavior is implementation dependent.

5.3.9.5 workingDirectory

This attribute specifies the directory where the job is executed. If this property is not set, the behavior is implementation dependent. This property value MUST be evaluated relative to the execution host's file system. The property value MAY contain the **HOME_DIRECTORY** or **PARAMETRIC_INDEX** constant values as placeholders. A **HOME_DIRECTORY** placeholder at the beginning denotes the remaining portion of the attribute value as a relative directory path resolved relative to the job users home directory at the execution host. The **PARAMETRIC_INDEX** placeholder MAY be used at any position within the property value in the case of parametric job templates and SHALL be substituted by the underlying DRM system with the parametric jobs' index.

The workingDirectory MUST be specified in a syntax that is common at the host

where the job is executed.

If this property is set and no placeholder is used, an absolute directory specification is expected.

If this property is set and the directory does not exist, the job MUST enter the state, JobProgramState.**FAILED**.

5.3.9.6 jobCategory

An implementation-defined string specifying how to resolve site-specific resources and/or policies. Site administrators MAY create a job category suitable for an application to be dispatched by the DRMS; the associated category name SHALL be specified as a job submission property. The DRMAA implementation MAY then use the category name to manage 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 application on top of it.

More information can be found in section 2.4.1 of the DRMAA 1.0 specification document.

5.3.9.7 nativeSpecification

An implementation-defined string that is passed by the end user to DRMAA to specify site-specific resources and/or policies.

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

The native specification MAY be used without the requirement to maintain job categories, and submit options MAY be specified directly.

More information can be found in section 2.4.2 of the DRMAA 1.0 specification document.

5.3.9.8 email

A list of email addresses that is used to report the job completion and status.

5.3.9.9 blockEmail

This boolean property decides whether the sending of email is blocked by default or not, regardless of the DRMS setting. If this property is `true`, the sending of email SHALL be blocked regardless of the DRMS setting. If this property is `false`, the sending of email SHALL be determined by the DRMS setting.

5.3.9.10 startTime

This property specifies the earliest time when the job MAY be eligible to be run.

5.3.9.11 jobName

A job name SHALL be comprised of alphanumeric and '_' characters. The DRMAA implementation MAY truncate any client-provided job name to an implementation-defined length that is at least 31 characters.

5.3.9.12 inputPath

Specifies the job's standard input as a path to a file. If this property is not explicitly set in the job template, the job is started with an empty input stream, unless the job category, native specification, or a DRMS setting causes a source for the input stream to be set. If this property is set, it specifies the network path for the job's input stream file in the form:

[hostname]:file_path

If the transferFiles job template property is supported and has a value where the FileTransferMode.**getTransferInputStream()** method returns `true`, the input file SHOULD be fetched by the underlying DRM system from the specified host, or from the submit host if no hostname was specified.

If the transferFiles job template attribute is not supported or its value's FileTransferMode.**getTransferInputStream()** method returns `false`, then the input file is always expected at the host where the job is executed, irrespective of whether a hostname was specified.

The **PARAMETRIC_INDEX** placeholder can be used at any position for parametric job templates and SHALL be substituted by the underlying DRM system with the parametric job's index.

A **HOME_DIRECTORY** placeholder at the beginning of the property value denotes the remaining portion as a relative file specification resolved relative to the job's user's home directory at the host where the file is located.

A **WORKING_DIRECTORY** placeholder at the beginning of the property value denotes the remaining portion as a relative file specification resolved relative to the job's working directory at the host where the file is located.

The inputPath property MUST be specified in a syntax that is common at the host where the file is located.

If this property is set, and the job was successfully submitted, and the file can't be read, the job enters the state, Session.**FAILED**.

5.3.9.13 outputPath

Specifies how to direct the job's standard output as a path to a file. If this property is not explicitly set in the job template, the destination of the job's output stream is not defined, unless the job category, native specification, or a DRMS setting causes a destination for the output stream to be set.. If this property is set, it specifies the network path of the job's output stream in the form:

[hostname]:file_path

If the transferFiles job template property is supported and its value's FileTransferMode.**getOutputStream()** method returns `true`, the output file SHALL be transferred by the underlying DRM system to the specified host or to the submit host if no hostname is specified.

If the transferFiles job template property is not supported or its value's FileTransferMode.**getOutputStream()** method returns `false`, the output file SHALL be kept at the host where the job is executed, irrespective of whether a hostname was specified.

All output sent to the job's standard output stream SHALL be appended to the named file. If the file does not exist at the time the job is executed, the file SHALL first be created.

The **PARAMETRIC_INDEX** placeholder can be used at any position with parametric job templates and SHALL be substituted by the underlying DRM system with the parametric job's index.

A **HOME_DIRECTORY** placeholder at the beginning denotes the remaining portion as a relative file specification resolved relative to the job users home directory at the host where the file is located.

A **WORKING_DIRECTORY** placeholder at the beginning denotes the remaining portion as a relative file specification resolved relative to the jobs working directory at the host where the file is located.

The outputPath MUST be specified in a syntax that is common at the host where the file is located.

If this property is set, and the job was successfully submitted, and the file can't be written before execution, the job MUST enter the state, Session.**FAILED**.

5.3.9.14 errorPath

Specifies how to direct the jobs' standard error to a file.

If not explicitly set in the job template, the job template, the destination of the job's error stream is not defined unless the job category, native specification, or a DRMS setting causes a destination for the error stream to be set. If this property is set, it specifies the network path of the job's error stream file in the form:

[hostname]:file_path

If the transferFiles job template property is supported and its value's FileTransferMode.**getTransferErrorStream()** method returns `true`, the error file SHALL be transferred by the underlying DRM system to the specified host or to the submit host if no hostname is specified.

If the transferFiles job template property is not supported or it's value's FileTransferMode.**getTransferErrorStream()** method returns `false`, the error file is always kept at the host where the job is executed irrespective of whether a hostname was specified.

All output sent to the job's standard error stream SHALL be appended to the named file. If the file does not exist at the time the job is executed, the file SHALL first be created.

The **PARAMETRIC_INDEX** placeholder can be used at any position for parametric job templates and SHALL be substituted by the underlying DRM system with the parametric jobs' index.

A **HOME_DIRECTORY** placeholder at the beginning denotes the remaining portion as a relative file specification, resolved relative to the job users home directory at the host where the file is located.

A **WORKING_DIRECTORY** placeholder at the beginning denotes the remaining portion as a relative file specification resolved relative to the jobs working directory at the host where the file is located.

The errorPath MUST be specified in a syntax that is common at the host where the file is located.

If this property is set, and the job was successfully submitted, and the file can't be written before execution, the job enters the state, Session.**FAILED**.

5.3.9.15 joinFiles

Specifies whether the error stream should be intermixed with the output stream. If not explicitly set in the job template, this property defaults to `false`. If this property is set to `true`, the underlying DRM system SHALL ignore the value of the errorPath property and intermix the standard error stream with the standard output stream as specified by the outputPath.

5.3.10 Optional Properties

5.3.10.1 transferFiles

Specifies how to transfer files between hosts. If this property is not explicitly set in the job template, the effect is the same as setting the property to a FileTransferMode instance with all properties set to `false`. This property works in conjunction with the inputPath, outputPath and errorPath properties.

This property is optional. An Implementation MUST throw an `UnsupportedAttributeException` if this attribute is not supported.

5.3.10.2 deadlineTime

Specifies a deadline after which the DRMS will abort or terminate the job.

This property is optional. An Implementation MUST throw an `UnsupportedAttributeException` if this attribute is not supported.

5.3.10.3 hardWallclockTimeLimit

This property specifies when the job's wall clock time limit has been exceeded. An implementation SHALL terminate a job that has exceeded its wall clock time limit. Suspended time SHALL also be counted towards this limit. This property's value MUST be given in seconds.

This property is optional. An Implementation MUST throw an `UnsupportedAttributeException` if this attribute is not supported.

5.3.11 softWallClockTimeLimit

This property specifies an estimate as to how much wall clock time the job will need to complete. Note that the suspended time is also counted towards this estimate. This attribute is intended to assist the scheduler. If the time specified is insufficient, the implementation MAY impose a scheduling penalty. This property's value MUST be given in seconds.

This property is optional. An Implementation MUST throw an `UnsupportedAttributeException` if this attribute is not supported.

5.3.12 hardRunDurationLimit

This property 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 property's value MUST be given in seconds.

This property is optional. An Implementation MUST throw an `UnsupportedAttributeException` if this attribute is not supported.

5.3.13 softRunDurationLimit

This property 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 is insufficient, the implementation MAY impose a scheduling penalty. This property's value MUST be given in seconds.

This property is optional. An Implementation MUST throw an `UnsupportedAttributeException` if this attribute is not supported.

5.4 The SimpleJobTemplate Class

A Java language binding implementation MUST provide a SimpleJobTemplate class which implements the JobTemplate interface and provides functional implementations for all methods defined by the JobTemplate interface. The SimpleJobTemplate class MUST also provide certain SPI fields and methods. See section 7.3. The format of the SimpleJobTemplate class is as follows:

```
public class org.ggf.drmaa.SimpleJobTemplate
       implements JobTemplate {
    protected java.lang.String remoteCommand;
    protected java.util.List args;
    protected int jobSubmissionState;
    protected java.util.Map jobEnvironment;
    protected java.lang.String workingDirectory;
    protected java.lang.String jobCategory;
    protected java.lang.String nativeSpecification;
    protected java.util.List email;
    protected boolean blockEmail;
    protected java.util.Date startTime;
    protected java.lang.String jobName;
    protected java.lang.String inputPath;
    protected java.lang.String outputPath;
    protected java.lang.String errorPath;
    protected boolean joinFiles;
    public JobTemplate();
    public void setRemoteCommand(java.lang.String command)
       throws org.ggf.drmaa.DrmaaException;
    public java.lang.String getRemoteCommand()
       throws org.ggf.drmaa.DrmaaException;
    public void setArgs(java.util.List args)
       throws org.ggf.drmaa.DrmaaException;
    public java.util.List getArgs()
       throws org.ggf.drmaa.DrmaaException;
    public void setJobSubmissionState(int state)
       throws org.ggf.drmaa.DrmaaException;
    public int getJobSubmissionState()
       throws org.ggf.drmaa.DrmaaException;
    public void setJobEnvironment(java.util.Map env)
       throws org.ggf.drmaa.DrmaaException;
    public java.util.Map getJobEnvironment()
```

```
    throws org.ggf.drmaa.DrmaaException;
public void setWorkingDirectory(java.lang.String wd)
    throws org.ggf.drmaa.DrmaaException;
public java.lang.String getWorkingDirectory()
    throws org.ggf.drmaa.DrmaaException;
public void setJobCategory(java.lang.String category)
    throws org.ggf.drmaa.DrmaaException;
public java.lang.String getJobCategory()
    throws org.ggf.drmaa.DrmaaException;
public void setNativeSpecification(java.lang.String spec)
    throws org.ggf.drmaa.DrmaaException;
public java.lang.String getNativeSpecification()
    throws org.ggf.drmaa.DrmaaException;
public void setEmail(java.util.Set email)
    throws org.ggf.drmaa.DrmaaException;
public java.util.Set getEmail()
    throws org.ggf.drmaa.DrmaaException;
public void setBlockEmail(boolean blockEmail)
    throws org.ggf.drmaa.DrmaaException;
public boolean getBlockEmail()
    throws org.ggf.drmaa.DrmaaException;
public void setStartTime(org.ggf.drmaa.PartialTimestamp startTime)
    throws org.ggf.drmaa.DrmaaException;
public org.ggf.drmaa.PartialTimestamp getStartTime()
    throws org.ggf.drmaa.DrmaaException;
public void setJobName(java.lang.String name)
    throws org.ggf.drmaa.DrmaaException;
public java.lang.String getJobName()
    throws org.ggf.drmaa.DrmaaException;
public void setInputPath(java.lang.String inputPath)
    throws org.ggf.drmaa.DrmaaException;
public java.lang.String getInputPath()
    throws org.ggf.drmaa.DrmaaException;
public void setOutputPath(java.lang.String outputPath)
    throws org.ggf.drmaa.DrmaaException;
public java.lang.String getOutputPath()
    throws org.ggf.drmaa.DrmaaException;
public void setErrorPath(java.lang.String errorPath)
    throws org.ggf.drmaa.DrmaaException;
public java.lang.String getErrorPath()
    throws org.ggf.drmaa.DrmaaException;
public void setJoinFiles(boolean joinFiles)
    throws org.ggf.drmaa.DrmaaException;
public boolean getJoinFiles()
    throws org.ggf.drmaa.DrmaaException;
public void setTransferFiles(org.ggf.drmaa.FileTransferMode mode)
    throws org.ggf.drmaa.DrmaaException;
public org.ggf.drmaa.FileTransferMode getTransferFiles()
    throws org.ggf.drmaa.DrmaaException;
public void setDeadlineTime(org.ggf.drmaa.PartialTimestamp
```

```
        deadline) throws org.ggf.drmaa.DrmaaException;
    public org.ggf.drmaa.PartialTimestamp getDeadlineTime()
        throws org.ggf.drmaa.DrmaaException;
    public void setHardWallclockTimeLimit(long limit)
        throws org.ggf.drmaa.DrmaaException;
    public long getHardWallclockTimeLimit()
        throws org.ggf.drmaa.DrmaaException;
    public void setSoftWallClockTimeLimit(long limit)
        throws org.ggf.drmaa.DrmaaException;
    public long getSoftWallClockTimeLimit()
        throws org.ggf.drmaa.DrmaaException;
    public void setHardRunDurationLimit(long limit)
        throws org.ggf.drmaa.DrmaaException;
    public long getHardRunDurationLimit()
        throws org.ggf.drmaa.DrmaaException;
    public void setSoftRunDurationLimit(long limit)
        throws org.ggf.drmaa.DrmaaException;
    public long getSoftRunDurationLimit()
        throws org.ggf.drmaa.DrmaaException;
    public java.util.Set getAttributeNames()
        throws org.ggf.drmaa.DrmaaException;
    public java.lang.String toString();
    public void modified();
    protected java.util.Set getOptionalAttributeNames();
}
```

The SimpleJobTemplate MUST provide concrete implementations for all abstract methods defined in the JobTemplate interface. The setters for all required properties MUST store copies of the property values in the appropriate member variables, and the getters for all required properties MUST provide copies of the stored property values. In the case of properties of type, java.util.Map, the associated getter MUST return a reference to the associated member variable wrapped in a call to java.util.Collections.**unmodifiableMap()**. In the case of properties of type, java.util.List, the associated getter MUST return a reference to the associated member variable wrapped in a call to java.util.Collections.**unmodifiableList()**. In the case of properties of type, java.util.Set, the associated getter MUST return a reference to the associated member variable wrapped in a call to java.util.Collections.**unmodifiableSet()**. The setters and getters for all optional attributes MUST throw an UnsupportedAttributeException. The **getAttributeNames()** method MUST return a list that consists of the names of all required properties and the names returned by calling the **getOptionalAttributeNames()** method. See section 5.3.

5.4.1 SimpleJobTemplate

The no-args constructor MUST instantiate a new SimpleJobTemplate instance with all property member variables set to default values. Unless otherwise specified, the default value for a property member variable is 0, `false`, or `null`, depending on its type.

Every call to a SimpleJobTemplate property setter SHALL mark the job template to indicate that its properties have been modified, causing the next call to the **toString**() method to recalculate its return value.

5.4.2 toString

This method SHALL return a string representation of the job template instance which includes the values for all properties which have been set. Unset property values MAY also be included in the string representation.

As long as no property value has been changed and no property getters throw an exception, this method SHALL buffer the resulting string to be returned by future calls to this method. If the job template has been marked to indicate that its properties have been modified, the return value SHALL be recalculated during the next call to this method.

5.4.2.1 Returns

This method SHALL return a string representation of the job template instance which includes the values for all properties which have been set. Unset property values MAY also be included in the string representation.

5.4.3 modified

This method SHALL mark the job template to indicate that its properties have been modified, causing the next call to the **toString**() method to recalculate its return value.

5.4.4 getOptionalAttributeNames

This method SHALL return an empty list.

5.4.4.1 Returns

This method SHALL return an empty list.

5.5 The JobInfo Interface

The information regarding a job's execution history is encapsulated by object instances that implement the JobInfo interface. Using the JobInfo interface, a DRMAA application can discover information about the resource usage and exit status of a job. The structure of the JobInfo interface is as follows:

```
public abstract interface org.ggf.drmaa.JobInfo {
    public abstract java.lang.String getJobId()
       throws org.ggf.drmaa.DrmaaException;
    public abstract java.util.Map getResourceUsage()
       throws org.ggf.drmaa.DrmaaException;
    public abstract boolean hasExited()
       throws org.ggf.drmaa.DrmaaException;
    public abstract int getExitStatus()
       throws org.ggf.drmaa.DrmaaException;
    public abstract boolean hasSignaled()
       throws org.ggf.drmaa.DrmaaException;
    public abstract java.lang.String getTerminatingSignal()
       throws org.ggf.drmaa.DrmaaException;
    public abstract boolean hasCoreDump()
       throws org.ggf.drmaa.DrmaaException;
```

```
    public abstract boolean wasAborted()
       throws org.ggf.drmaa.DrmaaException;
}
```

All methods of the JobInfo interface MAY raise the following exceptions in addition to any explicitly listed in the method descriptions below:

- java.lang.OutOfMemoryError – as described in the Java Language Specification. This exception replaces the OutOfMemoryException from the DRMAA IDL specification.
- DrmCommunicationException – The DRMS could not be contacted.
- AuthorizationException – the session owner does not have permission to perform the chosen operation.
- InternalException – due to an error in the DRMAA implementation, the chosen operation could not be performed.

5.5.1 getJobId

This method SHALL return the identifier for the completed job.

5.5.1.1 Returns

This method SHALL return the identifier for the completed job.

5.5.2 getResourceUsage

This method SHALL return the completed job's resource usage data. If the job did not produce resource usage data, this method SHALL return null. Please refer also to GFD.022 section 3.1.3 for more information about resource usage data semantics.

5.5.2.1 Returns

This method SHALL return the completed job's resource usage data or null if there is no resource usage data.

5.5.3 hasExited

This method SHALL return `true` if the job terminated normally. A return value of `false` MAY 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 the **getExitStatus()** method SHALL NOT provide exit status information. A return value of `true` indicates more detailed diagnosis can be retrieved by means of the **getExitStatus()** method.

5.5.3.1 Returns

This method SHALL return a boolean indicating whether the job has exited.

5.5.4 getExitStatus

If **hasExited()** returns `true`, this function SHALL return the operating system exit code of the job.

5.5.4.1 Returns

This method SHALL return the exit code of the job.

5.5.4.2 Throws

`DrmaaException` - MAY be be one of the following:

- java.lang.IllegalStateException – no exit state information is available.

5.5.5 hasSignaled

This method SHALL return `true` if the job terminated due to the receipt of a signal. A return value of `false` MAY 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 the **getTerminatingSignal()** method SHALL NOT provide signal information.

5.5.5.1 Returns

This method SHALL return a boolean indicating whether the job terminated due to a signal.

5.5.6 getTerminatingSignal

If the **hasSignaled()** method returns `true`, this method SHALL return a representation of the signal that caused the termination of the job. For signals declared by POSIX, the symbolic names SHALL be returned (e.g., SIGABRT, SIGALRM). For signals not declared by POSIX, a DRM-dependent string SHALL be returned.

5.5.6.1 Returns

This method SHALL return the name of the terminating signal.

5.5.6.2 Throws

`DrmaaException` - MAY be be one of the following:

- java.lang.IllegalStateException – the job did not terminate due to the receipt of a signal.

5.5.7 hasCoreDump

If the **hasSignaled()** method returns `true`, this method SHALL return `true` if a core image of the terminated job was created.

5.5.7.1 Returns

This method SHALL return a boolean indicating whether a core image of the terminated job was created.

5.5.7.2 Throws

`DrmaaException` - MAY be be one of the following:

- java.lang.IllegalStateException – the job did not terminate due to the receipt of a signal.

5.5.8 wasAborted

This method SHALL return `true` if the job ended before entering the running state.

5.5.8.1 Returns

This method SHALL return a boolean indicating whether the job ended before entering the running state.

5.6 The PartialTimestamp Class

The PartialTimestamp class is used by JobTemplate interface instances to represent partially specified time stamps, as required by the Distributed Resource Management Application API Specification 1.0. The PartialTimestamp class SHALL inherit all of its methods from the java.util.Calendar class, overriding the abstract methods to implement DRMAA-specific behavior. Except as noted below, the PartialTimestamp class SHALL behave like the java.util.GregorianCalendar class. For additional information, see the JavaDoc documentation for the java.util.Calendar class and the java.util.GregorianCalendar class.

Unlike java.util.Calendar, the PartialTimestamp class MUST not assume any default values for fields until they have been explicitly set. If the PartialTimestamp class is resolved to a concrete time (via the the java.util.Calendar.**getTime()** or java.util.Calendar.**getTimeInMillis()** method) before all fields are set, the unset fields SHALL be filled in using the current time in such a way that the resulting concrete time is the soonest possible time which agrees with the set fields and is not in the past. A PartialTimestamp object MAY be resolved to a concrete time any number of times. Each resolution will result in a concrete time that meets the above criteria for the point in time at which the resolution took place.

The resolving of partial time information MUST be performed according to the following rules:

- If the optional **ZONE_OFFSET** is not specified, the offset associated with the local timezone SHALL be used.
- If the **SECOND** field is not specified, then it SHALL be treated as zero.
- If neither the **DAY_OF_MONTH** nor the **DAY_OF_WEEK** nor the **DAY_OF_YEAR** field is specified, the current day SHALL be used unless the specified hour, minute and second has already elapsed, in which case the next day SHALL be used.
- If the **MONTH** field is not specified, the current month SHALL be used unless the specified day, hour, minute and second has already elapsed, in which case the next month SHALL be used.
- If the **YEAR** field is not specified, the current year SHALL be used unless the specified month, day, hour, minute and second has already elapsed, in which case the next year SHALL be used.
- If the **CENTURY** field is not specified, the current century SHALL be used unless the specified year, month, day, hour, minute and second has already elapsed, in which case the next century SHALL be used.

The structure of the PartialTimestamp class is as follows:

```
public class org.ggf.drmaa.PartialTimestamp
       extends java.util.Calendar {
    public static final int CENTURY;
    public static final int UNSET;
    public int getModifier(int field);
    public void setModifier(int field, int value);
}
```

5.6.1 CENTURY

The **CENTURY** constant replaces the java.util.Calendar.**ERA** constant. In a DRMAA partial time stamp, the time represented MUST always be after the beginning of the epoch, i.e. Jan 1st, 1970. Therefore, the **ERA** constant has no meaning. Instead, the **CENTURY** constant SHALL be used to represent all but the last two digits of the year. The last two digits of the year SHALL be represented by the java.util.Calendar.**YEAR** constant. This separation of the full year is required by the Distributed Resource Management Application API Specification 1.0.

5.6.2 UNSET

The **UNSET** constant is the value which MUST be returned by the java.util.Calendar.**get()** method for a field that has not been explicitly assigned a value.

5.6.3 getModifier

The **getModifier()** method SHALL return any modifiers which have been set for a field. Modifiers are set either by calling the **setModifier()** method, or as a side effect of the java.util.Calendar.**add()** method. Any time that a field in incremented, causing that field to roll over (for example, incrementing MONTH from 11 to 0), if the next highest order field is unset, a +1 modifier SHALL be added to the unset field. Any time that a field is decremented, causing that field to roll over (for example, decrementing MONTH from 0 to 11), if the next highest order field is unset, a -1 modifier SHALL be added to the unset field. In both cases, if the next highest order field is set, that field is incremented of decremented appropriately, as would normally happen with a java.util.Calendar instance.

5.6.4 setModifier

The **setModifier()** method allows modifiers to be set for fields. Such modifiers will be added to those fields' values. If a modifier is set for a field which has already been assigned a value, the modifier is simply added to that field's value. If a modifier is set for a field which has not been assigned a value, the modifier is applied to that field's value *after* the partial time stamp has been resolved to a concrete time via the java.util.Calendar.**getTime()** or java.util.Calendar.**getTimeInMillis()** method.

5.7 The PartialTimestampFormat Class

In order to translate a PartialTimestamp object to or from a String, the PartialTimestampFormat class is used. In order for a PartialTimestampFormat object to interpret a String, the String must be in the format described in the Distributed Resource Management Application API Specification 1.0. Namely, the value of the String must be of the form: `[[[[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 [0,99]
- 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 [0,59]

Strings not adhering to this format will cause a java.text.ParseException to be thrown.

The structure of the PartialTimestampFormat class is as follows:

```
public class org.ggf.drmaa.PartialTimestampFormat
       extends java.text.Format {
    public org.ggf.drmaa.PartialTimestampFormat();
    public java.lang.StringBuffer format(java.lang.Object obj,
       java.lang.StringBuffer stringBuffer,
       java.text.FieldPosition fieldPosition);
    public java.lang.StringBuffer format
       (org.ggf.drmaa.PartialTimestamp obj,
       java.lang.StringBuffer stringBuffer,
       java.text.FieldPosition fieldPosition);
    public java.lang.String format(org.ggf.drmaa.PartialTimestamp obj);
    public org.ggf.drmaa.PartialTimestamp parse
       (java.lang.String string) throws java.text.ParseException;
    public org.ggf.drmaa.PartialTimestamp parse
       (java.lang.String string,
        java.text.ParsePosition parsePosition);
    public java.lang.Object parseObject(java.lang.String string,
       java.text.ParsePosition parsePosition);
}
```

For additional information, see the JavaDoc documentation for the java.text.Format class.

5.7.1 format (Object, StringBuffer, FieldPosition)

This method MUST translate the PartialTimestamp instance into a DRMAA format time string and append the string to the given java.util.StringBuffer. Since the PartialTimestampFormat class doesn't use fields, the fieldPosition parameter SHALL be ignored. This method is equivalent to java.lang.StringBuffer.**append**(PartialTimestampFormat.**format**(Object)).

In order for this parsing operation to be performed, the PartialTimestamp instance must have no unset field of a lower order than the highest order set field, with the exception of the second and

zone offset fields. For example, if the **YEAR** is set, the **MONTH**, a day field, the **HOUR**, and the **MINUTE** must also be set for this operation to be performed. Failure to meet this criterion MUST result in a java.text.ParseException being thrown. **SECONDS** and **ZONE_OFFSET** are always optional.

5.7.1.1 Parameters

`obj` - the object to format.
`stringBuffer` - the StringBuffer to which to append the results.
`fieldPosition` – ignored.

5.7.1.2 Returns

This method MUST return a reference to the *stringBuffer* parameter.

5.7.2 format (PartialTimestamp, StringBuffer, FieldPosition)

This method MUST translate the PartialTimestamp instance into a DRMAA format time string and append the string to the given java.util.StringBuffer. Since the PartialTimestampFormat class doesn't use fields, the fieldPosition parameter SHALL be ignored. This method is equivalent to java.lang.StringBuffer.**append**(PartialTimestampFormat.**format**(Object)).

In order for this parsing operation to be performed, the PartialTimestamp instance must have no unset field of a lower order than the highest order set field, with the exception of the second and zone offset fields. For example, if the **YEAR** is set, the **MONTH**, a day field, the **HOUR**, and the **MINUTE** must also be set for this operation to be performed. Failure to meet this criterion MUST result in a java.text.ParseException being thrown. **SECONDS** and **ZONE_OFFSET** are always optional.

5.7.2.1 Parameters

`obj` - the object to format.
`stringBuffer` - the StringBuffer to which to append the results.
`fieldPosition` – ignored.

5.7.2.2 Returns

This method MUST return a reference to the *stringBuffer* parameter.

5.7.3 format (PartialTimestamp)

This method MUST translate the PartialTimestamp instance into a DRMAA format time string. This method is equivalent to PartialTimestampFormat.**format**(obj, new java.util.**StringBuffer**(), new java.text.**FieldPosition**(0)).**toString**().

In order for this parsing operation to be performed, the PartialTimestamp instance must have no unset field of a lower order than the highest order set field, with the exception of the second and zone offset fields. For example, if the **YEAR** is set, the **MONTH**, a day field, the **HOUR**, and the **MINUTE** must also be set for this operation to be performed. Failure to meet this criterion MUST result in a java.text.ParseException being thrown. **SECONDS** and **ZONE_OFFSET** are always optional.

5.7.3.1 Parameters

`obj` - the object to format

5.7.3.2 Returns

This method MUST returns the DRMAA format time string.

5.7.4 parse (String)

This method MUST translate a DRMAA format time string into a PartialTimestamp instance. This method SHALL parse as far into the string as possible. If this method encounters unparsable text after successfully parsing the **HOUR_OF_DAY** and **MINUTE** fields, it will stop and will not throw a java.text.ParseException.

5.7.4.1 Parameters

`str` - a DRMAA format time string

5.7.4.2 Returns

This method MUST return an appropriate PartialTimestamp instance.

5.7.4.3 Throws

`java.text.ParseException` - thrown if the string is not parsable.

5.7.5 parse (String, ParsePosition)

This method translates a DRMAA specified time string into a PartialTimestamp object. This method SHALL parse as far into the string as possible. Upon completion, the parse position object SHALL contain the index of the last character parsed.

5.7.5.1 Parameters

`str` - a DRMAA specified time string.
`parsePosition` - the parse position object.

5.7.5.2 Returns

This method MUST return an appropriate PartialTimestamp instance.

5.7.6 parseObject

This method MUST translate a DRMAA specified time string into a PartialTimestamp instance. This method SHALL parse as far into the string as possible. Upon completion, the parse position object SHALL contain the index of the last character parsed.

5.7.6.1 Parameters

`str` - a DRMAA specified time string.
`parsePosition` - the parse position object.

5.7.6.2 Returns

This method MUST return an appropriate PartialTimestamp instance.

5.8 The FileTransferMode Class

The FileTransferMode class is used by a JobTemplate instance to indicate the value for the transferFiles property. The class has three properties which determine which streams will be staged in or out. The structure of the FileTransferMode class is as follows:

```
public class org.ggf.drmaa.FileTransferMode
       implements java.io.Serializable, java.lang.Cloneable {
    public org.ggf.drmaa.FileTransferMode();
    public org.ggf.drmaa.FileTransferMode(boolean transferInputStream,
       boolean transferOutputStream, boolean transferErrorStream);
    public void setTransferInputStream(boolean transferInputStream);
    public boolean getTransferInputStream();
    public void setTransferOutputStream(boolean transferOutputStream);
    public boolean getTransferOutputStream();
    public void setTransferErrorStream(boolean transferErrorStream);
    public boolean getTransferErrorStream();
}
```

5.8.1 FileTransferMode()

The no-args constructor SHALL initialize all three properties' values to false.

5.8.2 FileTransferMode(boolean, boolean, boolean)

This constructor SHALL initialize all three properties' values to the values specified in the parameters.

5.8.2.1 Parameters

`transferInputStream` - whether to transfer input stream files
`transferOutputStream` - whether to transfer output stream files
`transferErrorStream` - whether to transfer error stream files

5.8.3 setTransferInputStream

This method SHALL set whether to transfer input stream files. If this property is set to `true`, the transferInputStream property of the corresponding job template SHALL be treated as the source from which input files should be copied.

5.8.3.1 Parameters

`transferInputStream` - whether to transfer input stream files

5.8.4 getTransferInputStream

This method SHALL return a boolean representing whether to transfer an input stream file.

5.8.4.1 Returns

This method SHALL return a boolean representing whether to transfer the input stream file.

5.8.5 setTransferOutputStream

This method SHALL set whether to transfer an output stream file. If this property is set to `true`, the transferOutputStream property of the corresponding job template SHALL be treated as the destination to which the output file should be copied.

5.8.5.1 Parameters

`transferOutputStream` - whether to transfer output stream files

5.8.6 getTransferOutputStream

This method SHALL return a boolean representing whether to transfer output stream files.

5.8.6.1 Returns

This method SHALL return a boolean representing whether to transfer output stream files.

5.8.7 setTransferErrorStream

This method SHALL set whether to transfer an error stream file. If this property is set to true, the transferErrorStream property of the corresponding job template SHALL be treated as the destination to which the error file should be copied.

5.8.7.1 Parameters

`transferErrorStream` - whether to transfer error stream files

5.8.8 getTransferErrorStream

This method SHALL return a boolean representing whether to transfer error stream files.

5.8.8.1 Returns

This method SHALL return a boolean representing whether to transfer error stream files.

5.9 The Version Class

The Version class is a holding class for the major and minor version numbers of the DRMAA implementation as returned by the Session.**getVersion()** method. The toString() method of a Version instance MUST return a String of the form, “<major>.<minor>”. The class structure follows:

```
public class org.ggf.drmaa.Version
       implements java.io.Serializable, java.lang.Cloneable,
```

```
       java.lang.Comparable {
    public org.ggf.drmaa.Version(int major, int minor);
    public int getMajor();
    public int getMinor();
    public int compareTo(Object obj);
}
```

5.9.1 Version

This constructor SHALL initialize the major and minor properties to the values specified in the parameters.

5.9.1.1 Parameters

`major` – The major version number
`minor` – The minor version number

5.9.2 getMajor

This method SHALL return the major version number.

5.9.2.1 Returns

This method SHALL return the major version number.

5.9.3 getMinor

This method SHALL return the minor version number.

5.9.3.1 Returns

This method SHALL return the minor version number.

5.10 Exceptions

All exceptions in the Java language binding MUST inherit from the DrmaaException class or the java.lang.RuntimeException class. The structure of DrmaaException is as follows:

```
public class org.ggf.drmaa.DrmaaException
    extends java.lang.Exception{
    public org.ggf.drmaa.DrmaaException();
    public org.ggf.drmaa.DrmaaException(java.lang.String message);
}
```

All exceptions under the DrmaaException class SHALL have the following structure:

```
public class org.ggf.drmaa.<NAME>Exception
    extends DrmaaException{
    public org.ggf.drmaa.<NAME>Exception();
    public org.ggf.drmaa.<NAME>Exception(java.lang.String message);
}
```

where <NAME> is the name of the exception.

5.10.1 The Exception Hierarchy

The DRMAA exception hierarchy is as follows:

- java.lang.Object
  - java.lang.Throwable
    - java.lang.Exception
      - *org.ggf.drmaa.DrmaaException*
        - org.ggf.drmaa.AlreadyActiveSessionException
        - org.ggf.drmaa.AuthorizationException
        - org.ggf.drmaa.DeniedByDrmException
        - org.ggf.drmaa.DrmCommunicationException
        - org.ggf.drmaa.DrmsExitException
        - org.ggf.drmaa.DrmsInitException
          - org.ggf.drmaa.InvalidContactStringException
            - org.ggf.drmaa.DefaultContactStringException
            - org.ggf.drmaa.NoDefaultContactStringSelectedException
        - org.ggf.drmaa.ExitTimeoutException
        - *org.ggf.drmaa.InconsistentStateException*
          - org.ggf.drmaa.HoldInconsistentStateException
          - org.ggf.drmaa.ReleaseInconsistentStateException
          - org.ggf.drmaa.ResumeInconsistentStateException
          - org.ggf.drmaa.SuspendInconsistentStateException
        - org.ggf.drmaa.InvalidAttributeValueException
          - org.ggf.drmaa.ConflictingAttributeValuesException
          - org.ggf.drmaa.InvalidAttributeFormatException
        - org.ggf.drmaa.InvalidJobException
        - org.ggf.drmaa.InvalidJobTemplateException
        - org.ggf.drmaa.NoActiveSessionException
        - org.ggf.drmaa.TryLaterException
        - org.ggf.drmaa.UnsupportedAttributeException
    - java.lang.RuntimeException
      - org.ggf.drmaa.InternalException

Exceptions listed in itallics exist only for behavior aggregation and SHALL be declared as abstract.

5.10.2 AlreadyActiveSessionException

Initialization failed due to existing DRMAA session.

5.10.3 AuthorizationException

The user is not authorized to perform the given operation.

5.10.4 ConflictingAttributeValuesException

The value of this attribute conflicts with one or more previously set properties.

5.10.5 DefaultContactStringException

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

5.10.6 DeniedByDrmException

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

5.10.7 DrmCommunicationException

Could not contact DRM system.

5.10.8 DrmsExitException

A problem was encountered while trying to exit the session.

5.10.9 DrmsInitException

A problem was encountered while trying to initialize the session.

5.10.10 ExitTimeoutException

The Session.**wait()** or Session.**synchronize()** call returned before all selected jobs entered the **DONE** or **FAILED** state.

5.10.11 HoldInconsistentStateException

The job cannot be moved to a **HOLD** state.

5.10.12 InternalException

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

5.10.13 InvalidAttributeFormatException

The format of the job template property value is improperly formatted, such as a badly formatted time stamp.

5.10.14 InvalidAttributeValueException

The value for the job template property is invalid.

5.10.15 InvalidContactStringException

The given contact string is not valid.

5.10.16 InvalidJobException

The job specified by the given job id does not exist.

5.10.17 InvalidJobTemplateException

The job template is not valid. It was either created incorrectly, i.e. not via Session.**createJobTemplate()**, or it has been deleted via the Session.**deleteJobTemplate()** method.

5.10.18 NoActiveSessionException

Method call failed because there is no active session.

5.10.19 NoDefaultContactStringSelectedException

No defaults contact string was provided or selected. DRMAA requires that the default contact string is selected when there is more than one default contact string due to multiple DRMAA implementations being present and available. (See 5.1.21.)

5.10.20 ReleaseInconsistentStateException

The job is not in a **HOLD** state, and hence cannot be released.

5.10.21 ResumeInconsistentStateException

The job is not in a suspended state (i.e. ***_SUSPENDED**), and hence cannot be resumed.

5.10.22 SuspendInconsistentStateException

The job is not in a state from which it can be suspended.

5.10.23 TryLaterException

The DRMS rejected the operation, possibly due to excessive load. A retry attempt may succeed, however.

5.10.24 UnsupportedAttributeException

The given job template property is not supported by the current DRMAA implementation.

5.10.25 Correlation to Error Codes

The following table shows how the error codes defined in the Distributed Resource Management Application API Specification 1.0 correlate to exceptions in the Distributed Resource Management Application API Java Language Binding and core Java language.

Error Code Name (DRMAA_ERRNO_...)	*Exception Name (org.ggf.drmaa....)*
SUCCESS	none
INTERNAL_ERROR	InternalException
DRM_COMMUNICATION_FAILURE	DrmCommunicationException
AUTH_FAILURE	AuthorizationException
INVALID_ARGUMENT	java.lang.IllegalArgumentException
NO_ACTIVE_SESSION	NoActiveSessionException
NO_MEMORY	java.lang.OutOfMemoryError
INVALID_CONTACT_STRING	InvalidContactStringException
DEFAULT_CONTACT_STRING_ERROR	DefaultContactStringException
NO_DEFAULT_CONTACT_STRING_SELECTED	NoDefaultContactStringException
DRMS_INIT_FAILED	DrmsInitException
ALREADY_ACTIVE_SESSION	AlreadyActiveSessionException
DRMS_EXIT_ERROR	DrmsExitException
INVALID_ATTRIBUTE_FORMAT	InvalidAttributeFormatException
INVALID_ATTRIBUTE_VALUE	InvalidAttributeValueException
CONFLICTING_ATTRIBUTE_VALUES	ConflictingAttributeValues
TRY_LATER	TryLaterException
DENIED_BY_DRM	DeniedByDrmException
INVALID_JOB	InvalidJobException
RESUME_INCONSISTENT_STATE	ResumeInconsistentStateException
SUSPEND_INCONSISTENT_STATE	SuspendInconsistentStateException
HOLD_INCONSISTENT_STATE	HoldInconsistentStateException
RELEASE_INCONSISTENT_STATE	ReleaseInconsistentStateException
EXIT_TIMEOUT	ExitTimeoutException
NO_RUSAGE	none

Table 3: Correlating Error Codes to Exceptions

The DRMAA_ERRNO_SUCCESS code does not need to be represented as an exception.

5.10.26 Correlation to IDL Exceptions

The following table shows how the error codes defined in the Distributed Resource Management Application API – IDL Binding 1.0 correlate to exceptions in the Distributed Resource Management Application API Java Language Binding and core Java language.

IDL Exception	*Exception Name (org.ggf.drmaa....)*
InvalidJobTemplateException	InvalidJobTemplateException
UnsupportedAttributeException	UnsupportedAttributeException
IllegalStateException	java.lang.IllegalStateException
NoMoreElementsException	none

Table 4: Correlating IDL Exceptions to Java Language Binding Exceptions

6. Java Language Binding Example

The Java application below is an example of an application that uses the DRMAA Java language binding interface. It illustrates submission of both single and bulk jobs. After submission the Session.**synchronize()** method is used to synchronize with all jobs. Finally the Session.**wait()** method is used to retrieve and print out information about the exit status of each job.

The path, which must be passed as argument to the program, is used directly for the job template JobTemplate remoteCommand property. The Java language binding example passes "5" as first argument to the job template args property. Assuming the example is run with the "/bin/sleep" UNIX command as an argument, and that a command "/bin/sleep" exists at the machine executing the job which behaves like the UNIX sleep(1) command, running this application with the parameter "/bin/sleep" will result in 32 jobs being run that sleep for 5 seconds each before finishing.

The source code follows:

```
import java.util.*;

import org.ggf.drmaa.*;

public class DrmaaExample {
  private static int NBULKS = 3;
  private static int JOB_CHUNK = 8;
  private Session session = null;

  public void main (String[] args) throws Exception {
    String jobPath = args[0];
    session = SessionFactory.getFactory().getSession();
    session.init("");

    JobTemplate jt = createJobTemplate(jobPath, 5, true);

    List allJobIds = new LinkedList();
    Set jobIds = null;
    boolean retry = true;

    for (int count = 0; count < NBULKS; count++) {
      do {
        try {
```

```
        jobIds = session.runBulkJobs(jt, 1, JOB_CHUNK, 1);
        retry = false;
      } catch (DRMCommunicationException e) {
        System.err.println("runBulkJobs() failed - retry: " +
                           e.getMessage());

        try {
          Thread.sleep(1000);
        } catch (InterruptedException e) {}
      }
    } while (retry);

    allJobIds.add(jobIds);
    System.out.println("submitted bulk job with jobids:");
    Iterator i = jobIds.iterator();

    while (i.hasNext()) {
      System.out.println ("\t \"" + i.next() + "\"");
    }
  }

  session.deleteJobTemplate(jt);

  /* submit some sequential jobs */
  jt = createJobTemplate(jobPath, 5, false);

  String jobId = null;
  retry = true;

  for (int count = 0; count < JOB_CHUNK; count++) {
    do {
      try {
        jobId = session.runJob(jt);
        retry = false;
      }
      catch (DRMCommunicationException e) {
        System.err.println("runJob() failed - retry: " +
                           e.getMessage ());

        try {
          Thread.sleep(1000);
        } catch (InterruptedException e) {}
      }
    } while (retry);

    System.out.println("\t \"" + jobId + "\"");
    jobIds.add(jobId);
  }

  session.deleteJobTemplate(jt);
```

```
    /* synchronize with all jobs */
    session.synchronize(allJobIds,
                        Session.TIMEOUT_WAIT_FOREVER,
                        false);
    System.out.println("synchronized with all jobs");

    /* wait all those jobs */
    Iterator i = allJobIds.iterator();

    while (i.hasNext()) {
      JobInfo status = null;
      String name = (String)i.next();

      status = session.wait(name, Session.TIMEOUT_WAIT_FOREVER);

      /* report how job finished */
      if (status.wasAborted()) {
        System.out.println("job \"" + name + "\" never ran");
      } else if (status.hasExited ()) {
        System.out.println("job \"" + name +
                           "\" finished regularly with exit status " +
                           status.getExitStatus());
      } else if (status.hasSignaled ()) {
        System.out.println("job \"" + name +
                           "\" finished due to signal " +
                           status.getTerminatingSignal ());
      } else {
        System.out.println("job \"" + name +
                           "\" finished with unclear conditions");
      }
    }
  }

  private JobTemplate createJobTemplate(String jobPath,
                                        int seconds,
                                        boolean isBulkJob)
                                        throws DrmaaException {
    JobTemplate jt = session.createJobTemplate();

    jt.setWorkingDirectory(JobTemplate.WORKING_DIRECTORY);
    jt.setRemoteCommand(jobPath);
    jt.setArgs(Collections.singletonList(Integer.toString(seconds)));
    jt.setJoinFiles(true);

    if (!isBulkJob) {
      jt.setOutputPath(":" + JobTemplate.HOME_DIRECTORY +
                       "/DRMAA_JOB");
    }
    else {
```

```
      jt.setOutputPath(":" + JobTemplate.HOME_DIRECTORY +
                       "/DRMAA_JOB" + JobTemplate.PARAMETRIC_INDEX);
    }

    return jt;
  }
}
```

7. Service Provider Interface

A Java language binding SHALL written to be extended by service providers to provide functionality specific to their DRM software. In order to provide a service provider implementation, a service provider must extend all abstract classes with custom implementations. Concrete service provider implementations of DRMAA classes MAY additionally implement the java.lang.Cloneable, java.io.Serializable, and/or java.lang.Comparable interfaces. Such DRMAA classes SHOULD also override the **equals**(), **hashCode**(), and **toString**() methods where appropriate. The abstract classes that must be overridden are as follows:

7.1 Session Interface

All methods of the Session interface must be implemented. For details, see section 5.1.

7.2 SessionFactory Class

The **getSession()** method of the SessionFactory class must be implemented. The implementation SHOULD create and return an appropriate Session implementation.

7.3 SimpleJobTemplate Class

An implementation MAY extend the SimpleJobTemplate class if needed, but is not required to do so. The **getOptionalAttributeNames()** method of the SimpleJobTemplate is used by the **getAttributeNames()** method to determine which optional attributes are supported by the implementation. This abstract method MUST be override in a service provider implementation and SHOULD return a List of Strings representing the names of the supported optional and implementation-specific properties.

Alternatively, a service provider implementation MAY provide a custom concrete implementation of the JobTemplate interface. In such a case, the Session implementation SHOULD throw an InvalidJobTemplateException when a method which accepts a JobTemplate type parameter is called with an instance of the SimpleJobTemplate class.

7.4 JobInfo Interface

A service provider implementation MUST provide a concrete implementation of the JobInfo interface.

8. 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. Also, the Java 2 Standard Edition Runtime Environment applies a fine-grained security model that can be assumed to provide some measure of protection at the point of execution.

9. Author Information

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

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, Inc.
17 Network Circle
Menlo Park, CA 94306

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

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

10. Intellectual Property Statement

The OGF 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 OGF Secretariat.

The OGF 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 OGF Executive Director.

11. Disclaimer

This document and the information contained herein is provided on an “As Is” basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

12. Full Copyright Notice