

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

Daniel Templeton, Sun Microsystems (maintainer)
Peter Tröger, Universität Potsdam
Roger Brobst, Cadence Design Systems
Andreas Haas*, Sun Microsystems
Hrabri Rajic*, Intel Americas Inc.
John Tollefsrud, Sun Microsystems
*co-chairs
August, 2006

Distributed Resource Management Application API Java™ Language Bindings 0.6.5

Status of This Memo

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

Copyright Notice

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

Table of Contents

1 Abstract.....	5
2 Overview.....	5
3 Design Decisions.....	5
3.1 Service Provider Interface.....	5
4 Relationship to Other DRMAA Specifications.....	5
5 The Java Language Binding API.....	5
5.1 The Session Interface.....	7
5.1.1 SUSPEND.....	8
5.1.2 RESUME.....	8
5.1.3 HOLD.....	8
5.1.4 RELEASE.....	8
5.1.5 TERMINATE.....	8
5.1.6 JOB_IDS_SESSION_ALL.....	9
5.1.7 JOB_IDS_SESSION_ANY.....	9
5.1.8 TIMEOUT_WAIT_FOREVER.....	9
5.1.9 TIMEOUT_NO_WAIT.....	9
5.1.10 UNDETERMINED.....	9
5.1.11 QUEUED_ACTIVE.....	9
5.1.12 SYSTEM_ON_HOLD.....	9
5.1.13 USER_ON_HOLD.....	9
5.1.14 USER_SYSTEM_ON_HOLD.....	9
5.1.15 RUNNING.....	10
5.1.16 SYSTEM_SUSPENDED.....	10
5.1.17 USER_SUSPENDED.....	10
5.1.18 USER_SYSTEM_SUSPENDED.....	10
5.1.19 DONE.....	10
5.1.20 FAILED.....	10
5.1.21 init.....	10
5.1.22 exit.....	11
5.1.23 createJobTemplate.....	11
5.1.24 deleteJobTemplate.....	11
5.1.25 runJob.....	12
5.1.26 runBulkJobs.....	12
5.1.27 control.....	13
5.1.28 synchronize.....	14
5.1.29 wait.....	14
5.1.30 getJobProgramStatus.....	15
5.1.31 getContact.....	16
5.1.32 getVersion.....	16
5.1.33 getDRMSInfo.....	16
5.1.34 getDRMAAImplementation.....	17
5.2 The SessionFactory Class.....	17
5.2.1 getFactory.....	17
5.2.2 getSession.....	17
5.3 The JobTemplate Class.....	17
5.3.1 HOLD_STATE.....	19
5.3.2 ACTIVE_STATE.....	19
5.3.3 HOME_DIRECTORY.....	19

5.3.4	WORKING_DIRECTORY.....	19
5.3.5	PARAMETRIC_INDEX.....	19
5.3.6	JobTemplate.....	19
5.3.7	getAttributeNames.....	20
5.3.8	Getters.....	20
5.3.9	Setters.....	20
5.4	The JobInfo Class.....	20
5.4.1	getJobId.....	21
5.4.2	getResourceUsage.....	21
5.4.3	ifExited.....	21
5.4.4	getExitStatus.....	21
5.4.5	ifSignaled.....	21
5.4.6	getTerminatingSignal.....	22
5.4.7	coreDump.....	22
5.4.8	ifAborted.....	22
5.5	The PartialTimestamp Class.....	22
5.5.1	CENTURY.....	23
5.5.2	UNSET.....	23
5.5.3	getModifier.....	23
5.5.4	setModifier.....	23
5.6	The PartialTimestampFormat Class.....	23
5.6.1	format (Object, StringBuffer, FieldPosition).....	24
5.6.2	format (PartialTimestamp, StringBuffer, FieldPosition).....	25
5.6.3	format (PartialTimestamp).....	25
5.6.4	parse (String).....	25
5.6.5	parse (String, ParsePosition).....	26
5.6.6	parseObject.....	26
5.7	The FileTransferMode Class.....	26
5.7.1	setTransferInputStream.....	27
5.7.2	getTransferInputStream.....	27
5.7.3	setTransferOutputStream.....	27
5.7.4	getTransferOutputStream.....	27
5.7.5	setTransferErrorStream.....	27
5.7.6	getTransferErrorStream.....	28
5.8	The Version Class.....	28
5.8.1	getMajor.....	28
5.8.2	getMinor.....	28
5.9	Exceptions.....	28
5.9.1	The Exception Hierarchy.....	29
5.9.2	AlreadyActiveSessionException.....	30
5.9.3	AuthorizationException.....	30
5.9.4	ConflictingAttributeValuesException.....	30
5.9.5	DefaultContactStringException.....	30
5.9.6	DeniedByDrmException.....	30
5.9.7	DrmCommunicationException.....	30
5.9.8	DrmsExitException.....	30
5.9.9	DrmsInitException.....	31
5.9.10	ExitTimeoutException.....	31
5.9.11	HoldInconsistentStateException.....	31
5.9.12	InternalException.....	31
5.9.13	InvalidAttributeFormatException.....	31
5.9.14	InvalidAttributeValueException.....	31

5.9.15 InvalidJobException.....	31
5.9.16 InvalidJobTemplateException.....	31
5.9.17 NoActiveSessionException.....	31
5.9.18 NoDefaultContactStringSelectedException.....	31
5.9.19 NoResourceUsageException.....	31
5.9.20 ReleaseInconsistentStateException.....	32
5.9.21 ResumeInconsistentStateException.....	32
5.9.22 SuspendInconsistentStateException.....	32
5.9.23 TryLaterException.....	32
5.9.24 UnsupportedAttributeException.....	32
5.9.25 Correlation to Error Codes.....	32
6 Java Language Binding Example.....	34
7 Service Provider Interface.....	36
7.1 Session Interface.....	36
7.2 SessionFactory Class.....	36
7.3 JobTemplate Class.....	36
7.3.1 Fields.....	37
7.3.2 getOptionalAttributeNames.....	37
7.4 JobInfo Class.....	37
7.5 Fields.....	37
7.6 JobInfo.....	38
7.6.1 Parameters.....	38
8 Security Considerations.....	38
9 Author Information.....	38
10 Intellectual Property Statement.....	39
11 Full Copyright Notice.....	39

Index of Tables

Table 1: DRMAA Attributes.....	6
Table 2: Correlating Error Codes to Exceptions.....	35

1. Abstract

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

2. Overview

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 implementable with any Java 2 Software Development Kit™ 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.

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 handling 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 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 the abstract classes in the Java language binding and providing implementations of the abstract methods, a vendor can tailor his implementation to his needs. The vendor implementation is 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 0.35. 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 specification. 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.

5. The Java Language Binding API

The DRMAA Interface Specification was written originally with a slant towards a C/C++ binding. As such, several aspects of the DRMAA interface needed to be altered slightly 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.

Additionally, some methods from the DRMAA specification fail to appear in 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 setter and getter for each DRMAA attribute. The advantage is that the property setters and getters 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 attributes, their corresponding Java property names, and their types.

DRMAA Attribute	Java Property	Type
<code>drmaa_remote_command</code>	<code>remoteCommand</code>	<code>java.lang.String</code>
<code>drmaa_v_argv</code>	<code>args</code>	<code>java.lang.String[]</code>
<code>drmaa_js_state</code>	<code>jobSubmissionState</code>	<code>int</code>
<code>drmaa_v_env</code>	<code>jobEnvironment</code>	<code>java.util.Map</code>
<code>drmaa_wd</code>	<code>workingDirectory</code>	<code>java.lang.String</code>
<code>drmaa_job_category</code>	<code>jobCategory</code>	<code>java.lang.String</code>
<code>drmaa_native_specification</code>	<code>nativeSpecification</code>	<code>java.lang.String</code>
<code>drmaa_v_email</code>	<code>email</code>	<code>java.lang.String[]</code>
<code>drmaa_block_email</code>	<code>BlockEmail</code>	<code>boolean</code>
<code>drmaa_start_time</code>	<code>startTime</code>	<code>org.ggf.drmaa.PartialTimestamp</code>
<code>drmaa_job_name</code>	<code>jobName</code>	<code>java.lang.String</code>
<code>drmaa_input_path</code>	<code>inputPath</code>	<code>java.lang.String</code>
<code>drmaa_output_path</code>	<code>outputPath</code>	<code>java.lang.String</code>
<code>drmaa_error_path</code>	<code>errorPath</code>	<code>java.lang.String</code>
<code>drmaa_join_files</code>	<code>joinFiles</code>	<code>boolean</code>
<code>drmaa_transfer_files</code>	<code>transferFiles</code>	<code>org.ggf.drmaa.FileTransferMode</code>
<code>drmaa_deadline_time</code>	<code>deadlineTime</code>	<code>org.ggf.drmaa.PartialTimestamp</code>
<code>drmaa_wct_hlimit</code>	<code>hardWallclockTimeLimit</code>	<code>long</code>
<code>drmaa_wct_slimit</code>	<code>softWallclockTimeLimit</code>	<code>long</code>
<code>drmaa_run_duration_hlimit</code>	<code>hardRunDurationLimit</code>	<code>long</code>
<code>drmaa_run_duration_slimit</code>	<code>softRunDurationLimit</code>	<code>long</code>

Table 1: DRMAA Attributes

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

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

All property setters and getters 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 is copied so that the data structure stored by the Java language binding is uncoupled from the data structure in the calling application.

Optional attributes are also represented by setters and getters. The Java binding implementation must provide implementations for setters and getters for all DRMAA attributes, both required and optional. The setter and getter implementations for optional attributes must throw `org.ggf.drmaa.UnsupportedAttributeException`. The service provider implementation should then override the setters and getters for supported optional attributes with methods that operate normally.

The `JobTemplate.getAttributeNames()` method returns the names of all properties supported by the service provider implementation, including required, optional, and implementation specific attributes. In order to get the values for supported attributes, such as in a property sheet, one should use introspection to call the appropriate setter and 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. It has the following structure:

```
public abstract interface org.ggf.drmaa.Session {
    public static final int SUSPEND
    public static final int RESUME
    public static final int HOLD
    public static final int RELEASE
    public static final int TERMINATE
    public static final java.lang.String JOB_IDS_SESSION_ALL
    public static final java.lang.String JOB_IDS_SESSION_ANY
    public static final long TIMEOUT_WAIT_FOREVER
    public static final long TIMEOUT_NO_WAIT
    public static final int UNDETERMINED
    public static final int QUEUED_ACTIVE
    public static final int SYSTEM_ON_HOLD
    public static final int USER_ON_HOLD
    public static final int USER_SYSTEM_ON_HOLD
    public static final int RUNNING
    public static final int SYSTEM_SUSPENDED
    public static final int USER_SUSPENDED
    public static final int USER_SYSTEM_SUSPENDED
    public static final int DONE
    public static final int FAILED
    public abstract void init(java.lang.String contactString)
        throws org.ggf.drmaa.DrmaaException
    public abstract void exit()
        throws org.ggf.drmaa.DrmaaException
    public abstract org.ggf.drmaa.JobTemplate
        createJobTemplate()
        throws org.ggf.drmaa.DrmaaException
    public abstract void
        deleteJobTemplate(org.ggf.drmaa.JobTemplate jobTemplate)
        throws org.ggf.drmaa.DrmaaException
    public abstract java.lang.String
        runJob(org.ggf.drmaa.JobTemplate jobTemplate)
```

```

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

```

All methods of the Session class MAY raise the following exceptions in addition to any explicitly listed:

- java.lang.OutOfMemoryError – as described in the Java Language 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.
- 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 the methods 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.

5.1.7 JOB_IDS_SESSION_ANY

The **JOB_IDS_SESSION_ANY** constant is used by the **wait()** method to indicate that the method 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 the methods 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 the methods 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, 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 is used 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 **getDrmSystem()**, **getDrmaImplementation()**, and **getContact()** methods of the Session class.

If *contact* is `null`, the default DRM system is used, provided there is only one DRMS available. If *contact* is `null`, 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 with throw an `AlreadyActiveSessionException`.

5.1.21.1 Parameters

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

5.1.21.2 Throws

`DrmaaException` - May 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` and the default contact string could not be used to connect to the DRMS.
- `NoDefaultContactStringSelectedException` – the `contact` parameter is `null` and more than one DRMAA implementation is available.

5.1.22 exit

The **`exit()`** is used to disengage from DRM and allow the DRMAA implementation to perform any necessary internal cleanup. This routine ends the current DRMAA session but doesn't affect any jobs (e.g., queued and running jobs remain queued and running). Any `JobTemplate` objects 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`.

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 job template. 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.

5.1.23.2 Throws

`DrmaaException` - May be one of the following:

- `DrmCommunicationException` – unable to communication with the DRMS

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 this job template. If a DRMAA implementation provides a finalizer method for the JobTemplate class, 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:

- DrmCommunicationException – unable to communication with the DRMS

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 with the **createJobTemplate()** method and have not previously been deleted with the **deleteJobTemplate()** method and MUST otherwise throw an InvalidJobTemplateException.

5.1.26.1 Parameters

beginIndex - the starting value for the loop index.

end Index- the terminating value for the loop index.

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

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

5.1.26.2 Returns

The **runJob()** 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 *jobId*. If *jobId* 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.

To avoid thread races in multithreaded applications, the DRMAA implementation user should explicitly synchronize this call with any other job submission calls or control calls that may change the number of remote jobs.

The legal values for *action* and their meanings are:

- **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 external 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

jobId - The id of the job to control.

action - 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 it can be resumed.
- *SuspendInconsistentStateException* – the job is not in a state from which it can be suspended.
- *HoldInconsistentStateException* – the job is not in a state from which it can be held.
- *ReleaseInconsistentStateException* – the job is not in a state from which it 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 *jobIds* have finished execution. If *jobIds* 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.

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 multithreaded applications, the DRMAA implementation user should explicitly synchronize this call with any other job submission calls or control calls that may change the number of remote jobs.

To prevent blocking indefinitely in this call, the caller may use a timeout specifying after 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 no result is available. If the call exits before the timeout has elapsed, all the jobs have been waited on or there was an interrupt. 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 will dispose of the job's data record at the end of the **synchroniize()** call. If set to `false`, the data record will be left for future access via the **wait()** method.

5.1.28.1 Parameters

jobIds - 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 routine will wait for any job submitted during this DRMAA session up to the moment **wait()** is called. This routine 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 if no result is available. Alternatively, a number of seconds may be specified to indicate how long to wait for a result to become available.

If the call exits before timeout, either the job has been waited on successfully or there was an interrupt. If the invocation exits on timeout, 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.) 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 job info object. 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 returns the resource usage and status information as a `JobInfo` object.

5.1.29.3 Throws

`DrmaaException` - May be one of the following:

- `NoResourceUsageDataException` – the resource usage information for the given job is unavailable.
- `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 `getJobProgramStatus`

The `getJobProgramStatus()` 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 active,
- **SYSTEM_ON_HOLD**: job is queued and in system hold,
- **USER_ON_HOLD**: job is queued and in user hold,
- **USER_SYSTEM_ON_HOLD**: job is queued and in user and system hold,
- **RUNNING**: job is running,

- **SYSTEM_SUSPENDED**: job is system suspended,
- **USER_SUSPENDED**: job is user suspended,
- **DONE**: job finished normally, and
- **FAILED**: job finished, but failed.

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.

5.1.30.1 Parameters

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

5.1.30.2 Returns

The **getJobProgramStatus()** 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 getContact

If called before **init()**, this method SHALL return a comma delimited String containing the contact Strings available from the default DRMAA implementation, one element per DRM system available. A contact string represents a specific installation of a specific DRM system, e.g. a 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.

5.1.31.1 Returns

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

5.1.32 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.

5.1.32.1 Returns

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

5.1.33 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.

5.1.33.1 Returns

This method SHALL return the DRM system identifier information.

5.1.34 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.

5.1.34.1 Returns

This method returns 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 com.sun.grid.drmaa.SessionFactory {
    public static com.sun.grid.drmaa.SessionFactory getFactory ()
    public abstract com.sun.grid.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 returns a Session object appropriate for the DRM in use.

5.2.1.1 Returns

This method returns a Session object appropriate for the DRM in use.

5.2.2 getSession

This method returns a SessionFactory object appropriate for the DRM in use.

5.2.2.1 Returns

This method returns a SessionFactory object appropriate for the DRM in use.

5.3 The JobTemplate Class

In order to define the attributes associated with a job, a DRMAA application uses the JobTemplate class. 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 then 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. The structure of the JobTemplate class is as follows:

```
public class org.ggf.drmaa.JobTemplate {
    public static final int HOLD_STATE
    public static final int ACTIVE_STATE
    public static final int HOME_DIRECTORY
    public static final int WORKING_DIRECTORY
    public static final int PARAMETRIC_INDEX
    public JobTemplate()
    public void setRemoteCommand(java.lang.String command)
        throws org.ggf.drmaa.InvalidAttributeException
    public java.lang.String getRemoteCommand()
    public void setArgs(java.util.List args)
        throws org.ggf.drmaa.InvalidAttributeException
    public java.util.List getArgs()
    public void setJobSubmissionState(int state)
        throws org.ggf.drmaa.InvalidAttributeException
    public int getJobSubmissionState()
    public void setJobEnvironment(java.util.Map env)
        throws org.ggf.drmaa.InvalidAttributeException
    public java.util.Map getJobEnvironment()
    public void setWorkingDirectory(java.lang.String wd)
        throws org.ggf.drmaa.InvalidAttributeException
    public java.lang.String getWorkingDirectory()
    public void setJobCategory(java.lang.String category)
        throws org.ggf.drmaa.InvalidAttributeException
    public java.lang.String getJobCategory()
    public void setNativeSpecification(java.lang.String spec)
        throws org.ggf.drmaa.InvalidAttributeException
    public java.lang.String getNativeSpecification()
    public void setEmail(java.util.List)
        throws org.ggf.drmaa.InvalidAttributeException
    public java.util.List getEmail()
    public void setBlockEmail(boolean blockEmail)
        throws org.ggf.drmaa.InvalidAttributeException
    public boolean getBlockEmail()
    public void setStartTime(org.ggf.drmaa.PartialTimestamp startTime)
```

```

        throws org.ggf.drmaa.InvalidAttributeException
    public org.ggf.drmaa.PartialTimestamp getStartTime()
    public void setJobName(java.lang.String name)
        throws org.ggf.drmaa.InvalidAttributeException
    public java.lang.String getJobName()
    public void setInputPath(java.lang.String inputPath)
        throws org.ggf.drmaa.InvalidAttributeException
    public java.lang.String getInputPath()
    public void setOutputPath(java.lang.String outputPath)
        throws org.ggf.drmaa.InvalidAttributeException
    public java.lang.String getOutputPath()
    public void setErrorPath(java.lang.String errorPath)
        throws org.ggf.drmaa.InvalidAttributeException
    public java.lang.String getErrorPath()
    public void setJoinFiles(boolean joinFiles)
        throws org.ggf.drmaa.InvalidAttributeException
    public boolean getJoinFiles()
    public void setTransferFiles(org.ggf.drmaa.FileTransferMode mode)
        throws org.ggf.drmaa.InvalidAttributeException
    public org.ggf.drmaa.FileTransferMode getTransferFiles()
    public void setDeadlineTime(org.ggf.drmaa.PartialTimestamp
        deadline) throws org.ggf.drmaa.InvalidAttributeException
    public org.ggf.drmaa.PartialTimestamp getDeadlineTime()
    public void setHardWallclockTimeLimit(long limit)
        throws org.ggf.drmaa.InvalidAttributeException
    public long getHardWallclockTimeLimit()
    public void setSoftWallClockTimeLimit(long limit)
        throws org.ggf.drmaa.InvalidAttributeException
    public long getSoftWallClockTimeLimit()
    public void setHardRunDurationLimit(long limit)
        throws org.ggf.drmaa.InvalidAttributeException
    public long getHardRunDurationLimit()
    public void setSoftRunDurationLimit(long limit)
        throws org.ggf.drmaa.InvalidAttributeException
    public long getSoftRunDurationLimit()
    public abstract java.util.List getAttributeNames()
        throws org.ggf.drmaa.DrmaaException
}

```

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.

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

The SPI implementation is also allowed to add implementation-specific attributes. The **JobTemplate.getAttributeNames()** method SHALL return the names of all job template attributes supported by the service provider implementation, including required, optional, and implementation specific attributes. In order to get the values for 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 may be queued but it is not 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 JobTemplate

The **JobTemplate()** constructor creates a new instance of a `JobTemplate`. In most cases, a DRMAA implementation will require that `JobTemplates` be created through the `Session.createJobTemplate()` method, however. In those cases, passing a `JobTemplate` created through the **JobTemplate()** constructor to the `Session.deleteJobTemplate()`, `Session.runJob()`, or `Session.runBulkJobs()` methods will result in an `InvalidJobTemplateException` being thrown.

5.3.7 getAttributeNames

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

5.3.7.1 Returns

This method returns the list of supported property names.

5.3.8 Getters

For each property listed in Table 1: DRMAA Attributes, the JobTemplate class has a corresponding getter. Each getter is of the form “public <propertyType> get<propertyName>().”

5.3.8.1 Returns

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

5.3.9 Setters

For each property listed in Table 1: DRMAA Attributes, the JobTemplate class has a corresponding setter. Each setter is of the form “public void set<propertyName>(<propertyType> value).” Setters for non-primitive, mutable properties store a copy of the new value instead of storing the original object.

5.3.9.1 Parameters

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

5.3.9.2 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.

5.3.10 Required Properties

5.3.10.1 `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.

5.3.10.2 `args`

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

5.3.10.3 `jobSubmissionState`

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

5.3.10.4 `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 behaviour is implementation dependent.

5.3.10.5 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 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 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 the property is set and no placeholder is used, an absolute directory specification is expected.

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

5.3.10.6 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.

5.3.10.7 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 job categories and native specification with the same job submission for policy specification. In this case, the DRMAA library is assumed to be capable of joining the outcome of the two policy sources in a reasonable way.

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.

5.3.10.8 email

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

5.3.10.9 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 will be blocked regardless of the DRMS setting. If this property is `false`, the sending of email will be determined by the DRMS setting.

5.3.10.10 startTime

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

5.3.10.11 jobName

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

5.3.10.12 inputPath

Specifies the job standard input as path to a file. Unless set elsewhere, if not explicitly set in the job template, the job is started with an empty input stream. If set, specifies the network path of the jobs input stream file of the form

[hostname]:file_path

When the `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.

When the `transferFiles` job template attribute is not supported or its values member `FileTransferMode.getTransferInputStream()` method returns `false`, then the input file is always expected at the host where the job is executed, irrespective of a possibly hostname 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 jobs' 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 users 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 jobs 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 set, and the file can't be read, the job enters the state `Session.FAILED`.

`outputPath`

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

`[hostname]:file_path`

When the `transferFiles` job template property is supported and its value's member `FileTransferMode.getTransferOutputStream()` 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.

When the `transferFiles` job template attribute is not supported or the `FileTransferMode.getTransferOutputStream()` method returns `false`, then the output file is always kept at the host where the job is executed irrespective of a possible `hostname` specified.

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 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 `outputPath` MUST be specified in a syntax that is common at the host where the file is located. If set and the file can't be written before execution the job enters the state `Session.FAILED`.

5.3.10.13 `errorPath`

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

If not explicitly set in the job template, the whereabouts of the jobs error stream is not defined. If set, specifies the network path of the jobs error stream file of the form

`[hostname]:file_path`

When the `transferFiles` job template property is supported and in its value the `FileTransferMode.getTransferErrorStream()` 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.

When the FileTransferMode.**getTransferErrorStream()** method returns *false*, the error file is always kept at the host where the job is executed irrespective of a possible hostname 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 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 set and the file can't be written before execution the job enters the state Session.**FAILED**.

5.3.10.14 joinFiles

Specifies if the error stream should be intermixed with the output stream. If not explicitly set in the job template the attribute defaults to *false*. If *true* is specified 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 with outputPath.

5.3.11 Optional Properties

5.3.11.1 transferFiles

Specifies how to transfer files between hosts. If not explicitly set in the job template, all members of the associated FileTransferMode instance are not set. This attribute works in conjunction with the inputPath, outputPath and errorPath properties.

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

5.3.11.2 deadlineTime

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

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

5.3.11.3 hardWallclockTimeLimit

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

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

5.3.12 softWallClockTimeLimit

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

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

5.3.13 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. The value MUST be given in seconds.

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

5.3.14 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 is optional. Implementations MUST throw an `UnsupportedAttributeException` if this attribute is not supported.

5.4 The JobInfo Class

The information regarding a job's execution is encapsulated in the `JobInfo` class. Via the `JobInfo` class a DRMAA application can discover information about the resource usage and exit status of a job. The structure of the `JobInfo` class is as follows:

```
public abstract class org.ggf.drmaa.JobInfo
    implements java.io.Serializable {
    public java.lang.String getJobId()
    public java.util.Map getResourceUsage()
    public abstract boolean hasExited()
    public abstract int getExitStatus()
    public abstract boolean hasSignaled()
    public abstract java.lang.String getTerminatingSignal()
    public abstract boolean hasCoreDump()
    public abstract boolean wasAborted()
}
```

5.4.1 getJobId

This method returns the identifier of the completed job.

5.4.1.1 Returns

This method returns the identifier of the completed job.

5.4.2 getResourceUsage

This method returns the completed job's resource usage data.

5.4.2.1 Returns

This method returns the completed job's resource usage data.

5.4.3 hasExited

This method SHALL contain `true` if the job terminated normally. `False` MAY also indicate that although the job has terminated normally, an exit status is not available, or that it is not known whether the job terminated normally. In both cases the **getExitStatus()** method SHALL NOT contain exit status information. `True` indicates more detailed diagnosis can be retrieved by means of the **getExitStatus()** method.

5.4.3.1 Returns

This method returns a boolean indicating whether the job has exited.

5.4.4 getExitStatus

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

5.4.4.1 Returns

This method returns the exit code for the job.

5.4.5 hasSignaled

This attribute SHALL contain `true` if the job terminated due to the receipt of a signal. `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.

5.4.5.1 Returns

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

5.4.6 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.

5.4.6.1 Returns

This method returns the name of the terminating signal.

5.4.7 hasCoreDump

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

5.4.7.1 Returns

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

5.4.8 wasAborted

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

5.4.8.1 Returns

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

5.5 The PartialTimestamp Class

The `PartialTimestamp` class is used by the `JobTemplate` class to represent partially specified time stamps, as required by the Distributed Resource Management Application API Specification 1.0. The `PartialTimestamp` class inherits all of its methods from the `java.util.Calendar` class, overriding the abstract methods to implement DRMAA-specific behavior. For additional information, see the JavaDoc documentation for the `java.util.Calendar` class.

Unlike `java.util.Calendar`, the `PartialTimestamp` class does 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 `java.util.Calendar.getTime()` or `java.util.Calendar.getTimeInMillis()` method) before all the fields are set, the unset fields will be filled in using the current time in such a way that the resulting concrete time is the soonest possible time which adheres to 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 structure of the `DrmaaCalendar` 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 void getModifier(int field)
    public void setModifier(int field, int value)
}
```

5.5.1 CENTURY

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

5.5.2 UNSET

The **UNSET** constant is the value returned by the `java.util.Calendar.get()` method before a field has been assigned a value.

5.5.3 getModifier

The **getModifier()** method returns any modifiers which have been set for a field. Modifiers are set either by the **setModifier()** method, or as a side effect of the `java.util.Calendar.add()` method.

5.5.4 setModifier

The **setModifier()** method allows modifiers to be set for fields which will be added to those field's values. If a modifier is set for a field which has been assigned a value, the modifier is simply added to the field's value. If a modifier is set for a field which has not been assigned a value, the modifier is applied to the 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.

5.6 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 be able to interpret a `String`, it 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 (century-1)
- YY is the last two digits of the year
- MM is the two digits of the month [01,12]
- DD is the two-digit day of the month [01,31]
- hh is the two-digit hour of the day [00,23]
- mm is the two-digit minute of the day [00,59]
- ss is the two-digit second of the minute [00,61]
- UU is the two-digit hours since (before) UTC
- uu is the two-digit minutes since (before) UTC

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.

5.6.1 **format (Object, StringBuffer, FieldPosition)**

This method translates the `PartialTimestamp` into a DRMAA specified time string and appends the string to the given `StringBuffer`. Since the `PartialTimestampFormat` class doesn't use fields, the `fieldPosition` parameter is ignored. This method is equivalent to `stringBuffer.append(format(obj))`.

In order to represent a `PartialTimestamp` object as a string, the `PartialTimestamp` object cannot have an unset field that is less significant than the most significant set field. That is to say that if **CENTURY** is set, **YEAR**, **MONTH**, and **DAY_OF_MONTH** must also be set. **SECONDS** and **ZONE_OFFSET** are always optional.

5.6.1.1 Parameters

`obj` - the object to format.

`stringBuffer` - the `StringBuffer` to which to append the results.

`fieldPosition` - ignored.

5.6.1.2 Returns

This method returns the `stringBuffer` parameter.

5.6.2 **format (PartialTimestamp, StringBuffer, FieldPosition)**

This method translates the `PartialTimestamp` into a DRMAA specified time string and appends the string to the given `StringBuffer`. Since the `PartialTimestampFormat` class doesn't use fields, the `fieldPosition` parameter is ignored. This method is equivalent to `stringBuffer.append(format(obj))`.

In order to represent a `PartialTimestamp` object as a string, the `PartialTimestamp` object cannot have an unset field that is less significant than the most significant set field. That is to say that if

CENTURY is set, **YEAR**, **MONTH**, and **DAY_OF_MONTH** must also be set. **SECONDS** and **ZONE_OFFSET** are always optional.

5.6.2.1 Parameters

`obj` - the object to format.

`stringBuffer` - the `StringBuffer` to which to append the results.

`fieldPosition` - ignored.

5.6.2.2 Returns

This method returns the `stringBuffer` parameter.

5.6.3 format (PartialTimestamp)

This method translates the `PartialTimestamp` into a DRMAA specified time string. This method is equivalent to `format (obj, new StringBuffer (), new FieldPosition (0)).toString ()`.

In order to represent a `PartialTimestamp` object as a string, the `PartialTimestamp` object cannot have an unset field that is less significant than the most significant set field. That is to say that if **CENTURY** is set, **YEAR**, **MONTH**, and **DAY_OF_MONTH** must also be set. **SECONDS** and **ZONE_OFFSET** are always optional.

5.6.3.1 Parameters

`obj` - the object to format

5.6.3.2 Returns

This method returns the DRMAA specified time string.

5.6.4 parse (String)

This method translates a DRMAA specified time string into a `PartialTimestamp` object. This method will parse as far as possible, but after successfully parsing the **HOURLY_OF_DAY** and **MINUTE** fields, if it encounters unparseable text, it will stop and will not throw a `java.text.ParseException`.

5.6.4.1 Parameters

`str` - a DRMAA specified time string

5.6.4.2 Returns

This method returns the corresponding `PartialTimestamp` object.

5.6.4.3 Throws

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

5.6.5 parse (String, ParsePosition)

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

5.6.5.1 Parameters

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

5.6.5.2 Returns

This method returns the corresponding PartialTimestamp object.

5.6.6 parseObject

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

5.6.6.1 Parameters

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

5.6.6.2 Returns

This method returns the corresponding PartialTimestamp object.

5.7 The FileTransferMode Class

The FileTransferMode class is used by the JobTemplate class to indicate the value for the transferFiles property. The class has three properties which determine which streams will be staged in or out. See the transferFiles property in the Distributed Resource Management Application API Specification 1.0 for more information. 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()
}
```

5.7.1 setTransferInputStream

This method sets whether to transfer input stream files. If this method 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.

5.7.1.1 Parameters

`transferInputStream` - whether to transfer input stream files

5.7.2 getTransferInputStream

This method returns a boolean representing whether to transfer input stream files.

5.7.2.1 Returns

This method returns a boolean representing whether to transfer input stream files.

5.7.3 setTransferOutputStream

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

5.7.3.1 Parameters

`transferOutputStream` - whether to transfer output stream files

5.7.4 getTransferOutputStream

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

5.7.4.1 Returns

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

5.7.5 setTransferErrorStream

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

5.7.5.1 Parameters

`transferErrorStream` - whether to transfer error stream files

5.7.6 getTransferErrorStream

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

5.7.6.1 Returns

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

5.8 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 `DrmaaSession.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 {
    public org.ggf.drmaa.Version(int major, int minor)
    public int getMajor()
    public int getMinor()
}
```

5.8.1 getMajor

This method the major version number.

5.8.1.1 Returns

This method the major version number.

5.8.2 getMinor

This method the minor version number.

5.8.2.1 Returns

This method the minor version number.

5.9 Exceptions

All exceptions in the Java language binding inherit from the `DrmaaException` or `DrmaaRuntimeException` classes . The structure of `DrmaaException` as as follows:

```
public class com.sun.grid.drmaa.DrmaaException
    extends java.lang.Exception{
    public com.sun.grid.drmaa.DrmaaException()
    public com.sun.grid.drmaa.DrmaaException(java.lang.String message)
}
```

The structure of `DrmaaRuntimeException` is as follows:

```
public class com.sun.grid.drmaa.DrmaaRuntimeException
    extends java.lang.RuntimeException{
    public com.sun.grid.drmaa.DrmaaRuntimeException()
    public com.sun.grid.drmaa.DrmaaRuntimeException(java.lang.String
```

```
message)
}
```

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

```
public class com.sun.grid.drmaa.<NAME>Exception
    extends DrmaaException{
    public com.sun.grid.drmaa.<NAME>Exception ()
    public com.sun.grid.drmaa.<NAME>Exception (java.lang.String message)
}
```

where `<NAME>` is the name of the exception, except for `NoResourceUsageException`. The structure of `NoResourceUsageException` is as follows:

```
public class com.sun.grid.drmaa.NoResourceUsageException
    extends DrmaaException{
    public com.sun.grid.drmaa.NoResourceUsageException ()
    public com.sun.grid.drmaa.NoResourceUsageException (
        java.lang.String message)
    public com.sun.grid.drmaa.NoResourceUsageException (
        java.lang.String message, java.lang.String jobId)
    public java.lang.String getJobId()
}
```

where the `jobId` attribute represents the id of the job for which no resource usage information is available.

All exceptions under the `DrmaaRuntimeException` class have the following structure:

```
public class com.sun.grid.drmaa.<NAME>Exception
    extends DrmaaRuntimeException{
    public com.sun.grid.drmaa.<NAME>Exception ()
    public com.sun.grid.drmaa.<NAME>Exception (java.lang.String message)
}
```

where `<NAME>` is the name of the exception.

5.9.1 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.DeniedByDrmException`
 - `org.ggf.drmaa.DrmCommunicationException`
 - `org.ggf.drmaa.DrmsExitException`
 - *`org.ggf.drmaa.InconsistentStateException`*
 - `org.ggf.drmaa.HoldInconsistentStateException`
 - `org.ggf.drmaa.ReleaseInconsistentStateException`
 - `org.ggf.drmaa.ResumeInconsistentStateException`
 - `org.ggf.drmaa.SuspendInconsistentStateException`
 - `org.ggf.drmaa.DrmsInitException`

- org.ggf.drmaa.InvalidContactStringException
 - org.ggf.drmaa.DefaultContactStringException
 - org.ggf.drmaa.NoDefaultContactStringSelectedException
- org.ggf.drmaa.InvalidJobException
- *org.ggf.drmaa.InvalidAttributeException*
 - org.ggf.drmaa.ConflictingAttributeValuesException
 - org.ggf.drmaa.InvalidAttributeFormatException
 - org.ggf.drmaa.InvalidAttributeValueException
- org.ggf.drmaa.NoResourceUsageException
- org.ggf.drmaa.ExitTimeoutException
- *org.ggf.drmaa.SessionException*
 - org.ggf.drmaa.NoActiveSessionException
 - org.ggf.drmaa.AlreadyActiveSessionException
- org.ggf.drmaa.TryLaterException
- java.lang.RuntimeException
 - *org.ggf.drmaa.DrmaaRuntimeException*
 - org.ggf.drmaa.AuthorizationException
 - org.ggf.drmaa.InternalException
 - org.ggf.drmaa.UnsupportedAttributeException
 - org.ggf.drmaa.InvalidJobTemplateException

Exceptions listed in italics exist only for behavior aggregation and must be declared as abstract.

5.9.2 AlreadyActiveSessionException

Initialization failed due to existing DRMAA session.

5.9.3 AuthorizationException

The user is not authorized to perform the given operation.

5.9.4 ConflictingAttributeValuesException

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

5.9.5 DefaultContactStringException

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

5.9.6 DeniedByDrmException

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

5.9.7 DrmCommunicationException

Could not contact DRM system.

5.9.8 DrmsExitException

A problem was encountered while trying to exit the session.

5.9.9 DrmsInitException

A problem was encountered while trying to initialize the session.

5.9.10 ExitTimeoutException

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

5.9.11 HoldInconsistentStateException

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

5.9.12 InternalException

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

5.9.13 InvalidAttributeFormatException

The format of the job template property value is invalid.

5.9.14 InvalidAttributeValueException

The value for the job template property is invalid.

5.9.15 InvalidJobException

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

5.9.16 InvalidJobTemplateException

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

5.9.17 NoActiveSessionException

Method call failed because there is no active session.

5.9.18 NoDefaultContactStringSelectedException

No default 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.

5.9.19 NoResourceUsageException

This exception is thrown by `Session.wait()` when a job has finished but no resource usage or exit status data could be provided. The `jobId` attribute indicates the id of the job for which this exception was thrown.

5.9.20 ReleaseInconsistentStateException

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

5.9.21 ResumeInconsistentStateException

The job is not in a ***_SUSPENDED** state, and hence cannot be resumed.

5.9.22 SuspendInconsistentStateException

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

5.9.23 TryLaterException

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

5.9.24 UnsupportedAttributeException

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

5.9.25 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 Bindings 0.5.1 and core Java language.

<i>Error Code Name (DRMAA_ERRNO_...)</i>	<i>Exception Name (org.ggf.drmaa....)</i>
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
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	NoResourceUsageException
none	InvalidJobTemplateException
none	UnsupportedAttributeException

Table 2: Correlating Error Codes to Exceptions

The DRMAA_ERRNO_SUCCESS code clearly does not need to be represented as an exception. The Java language binding introduces two new exceptions which have no error code correlaries. The InvalidJobTemplateException is an unchecked exception and is used to indicate that the job template object currently being used is not valid. This may be, for example, because it has already been deleted via Session.**deleteJobTemplate()**. The UnsupportedAttributeException is also an unchecked exception and is used to indicated that for the current DRMAA implementation the given property is unsupported.

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 `DrmaaSession.synchronize()` is used to synchronize with all jobs to finish. Finally `DrmaaSession.wait()` 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 directly used for the job template `JobTemplate.REMOTE_COMMAND` attribute. The Java language binding example passes "5" as first argument to the job template `JobTemplate.INPUT_PARAMETERS` attribute. Assuming the example is run under "/bin/sleep" unix command and that a command "/bin/sleep" exists at the remote machine 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 DrmaaSession session = null;

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

        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 ());

                    Thread.sleep (1000);
                }
            }
            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 ("runBulkJobs() failed - retry: " +
                e.getMessage ());

            Thread.sleep (1000);
        }
    }
    while (retry);

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

session.deleteJobTemplate (jt);

/* synchronize with all jobs */
session.synchronize (allJobIds,
    DrmaaSession.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;

    status = session.wait ((String)i.next (),
        DrmaaSession.TIMEOUT_WAIT_FOREVER);

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

```

```

        else {
            System.out.println ("job \"" + i.next () +
                               "\" finished with unclear conditions");
        }
    }
}

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

    jt.setWorkingDirectory (DrmaaSession.WORKING_DIRECTORY);
    jt.setRemoteCommand (jobPath);
    jt.setInputParameters (new String[] {
        Integer.toString (seconds);
    })

    jt.setJoinFiles (true);

    if (!isBulkJob) {
        jt.setOutputPath (DrmaaSession.HOME_DIRECTORY + "/DRMAA_JOB");
    }
    else {
        jt.setOutputPath (DrmaaSession.HOME_DIRECTORY +
                          "/DRMAA_JOB$drmaa_incr_ph$");
    }

    return jt;
}
}

```

7. Service Provider Interface

The Java language binding is 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 the following classes with custom implementations:

7.1 Session Interface

All methods of the Session interface must be implemented. For details, see section 4.

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 JobTemplate Class

An implementation may extend the JobTemplate class if needed, but is not required to do so. If the JobTemplate class is extended by the implementation, the JobTemplate implementation has access to the following protected fields and methods:

```

public abstract class org.ggf.drmaa.JobTemplate {
    protected java.lang.String remoteCommand

```

```

protected java.util.List args
protected int jobSubmissionState
protected java.util.Properties 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
protected java.util.List getOptionalAttributeNames ()
}

```

7.3.1 Fields

The protected fields of the JobTemplate class each correspond to the JobTemplate property of the same name. The default getter for each of the represented properties stores the property value in the corresponding field.

7.3.2 getOptionalAttributeNames

The `getOptionalAttributeNames()` method is used by the `getAttributeNames()` method to determine which optional attributes are supported by the implementation. This method should return a List of Strings representing the names of the supported optional and implementation-specific attributes.

7.4 JobInfo Class

All abstract methods of the JobInfo class must be implemented. See section 7 for details. Additionally, the JobInfo implementation has access to the following protected fields and constructors:

```

public abstract class org.ggf.drmaa.JobInfo
    implements java.io.Serializable {
    protected java.lang.String jobId
    protected int status
    protected java.util.Map resourceUsage
    protected org.ggf.drmaa.JobInfo (java.lang.String jobName,
        int statusCode,
        java.util.Map resourceUsage)
}

```

7.5 Fields

The `jobId` and `resourceUsage` fields correspond to the JobInfo properties of the same names. The `status` field is used to store the status code passed to the constructor.

1.1 JobInfo

This constructor stores the job name, status code, and resource usage information in the protected fields.

7.5.1 Parameters

`jobId` - the id of the job.

`status` - the status code of the job.

`resourceUsage` - the resource usage data for the job.

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 GmbH
Dr.-Leo-Ritter-Str. 7
D-93049 Regensburg
Germany

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
Universität Potsdam
Am Neuen Palais 10
D-14469 Potsdam
Germany

10. Intellectual Property Statement

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

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

11. Full Copyright Notice

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

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

The limited permissions granted above are perpetual and will not be revoked by the GGF or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE GLOBAL GRID FORUM DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE."