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

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

Distributed Resource Management Application API 1.0 - IDL Specification

Status of This Document

This document provides information to the Grid community. Distribution is unlimited.

Abstract

This document describes the common base for the Distributed Resource Management Application API (DRMAA) bindings for procedural and object-oriented languages. The document reflects the original semantics from the DRMAA 1.0 OGF recommendation document.

Table of Contents

1 INTRODUCTION 4

1.1 NOTATIONAL CONVENTIONS 4

2 GENERAL CONCEPTS 4

2.1 DESIGN DECISIONS 4
2.2 IDL LANGUAGE MAPPING 5

3 THE DRMAA API MODULE 6

4 DATA TYPES 7

4.1 JOBCONTROLACTION ENUMERATION 7
4.2 JOBPROGRAMSTATE ENUMERATION 7
4.3 JOBSUBMISSIONSTATE ENUMERATION 8
4.4 FILETRANSFERMODE VALUE TYPE 8
4.5 VERSION VALUE TYPE 9

5 EXCEPTIONS 9

5.1 ALREADYACTIVESESSIONEXCEPTION 10
5.2 AUTHORIZATIONEXCEPTION 10
5.3 CONFLICTINGATTRIBUTEVALUESEXCEPTION 10
5.4 DEFAULTCONTACTSTRINGEXCEPTION 10
5.5 DENIEDBYDRMEXCEPTION 10
5.6 DRMCOMMUNICATIONEXCEPTION 11
5.7 DRMSEXITEXCEPTION 11
5.8 DRMSINITEXCEPTION 11
5.9 EXITTIMEOUTEXCEPTION 11
5.10 HOLDINCONSISTENTSTATEEXCEPTION 11
5.11 INTERNALEXCEPTION 11
5.12 INVALIDARGUMENTEXCEPTION 11
5.13 INVALIDATTRIBUTEFORMATEXCEPTION 11
5.14 INVALIDATTRIBUTEVALUEEXCEPTION 11
5.15 INVALIDCONTACTSTRINGEXCEPTION 11
5.16 INVALIDJOBEXCEPTION 12
5.17 INVALIDJOBTEMPLATEEXCEPTION 12
5.18 NOACTIVESESSIONEXCEPTION 12
5.19 NODEFAULTCONTACTSTRINGSELECTEDEXCEPTION 12
5.20 OUTOFMEMORYEXCEPTION 12
5.21 RELEASEINCONSISTENTSTATEEXCEPTION 12
5.22 RESUMEINCONSISTENTSTATEEXCEPTION 12
5.23 SUSPENDINCONSISTENTSTATEEXCEPTION 12
5.24 TRYLATEREXCEPTION 12
5.25 UNSUPPORTEDATTRIBUTEEXCEPTION 12
5.26 ILLEGALSTATEEXCEPTION 13

6 THE PARTIALTIMESTAMP 13

7 JOBINFO INTERFACE 14

7.1 JOBID 15
7.2 RESOURCEUSAGE 15
7.3 HASEXITED 15
7.4 EXITSTATUS 15
7.5 HASSIGNALED 15

7.6 TERMINATINGSIGNAL 15
7.7 HASCOREDUMP 16
7.8 WASABORTED 16

8 JOBTEMPLATE INTERFACE 16

8.2 ACCESSING IMPLEMENTATION-SPECIFIC ATTRIBUTES 19
8.3 CONSTANTS 20
8.4 REMOTECOMMAND 20
8.5 ARGS 20
8.6 JOBSUBMISSIONSTATE 20
8.7 JOBENVIRONMENT 20
8.8 WORKINGDIRECTORY 20
8.9 JOBCATEGORY 21
8.10 NATIVESPECIFICATION 21
8.11 EMAIL 21
8.12 BLOCKEMAIL 21
8.13 STARTTIME 21
8.14 JOBNAME 22
8.15 INPUTPATH 22
8.16 OUTPUTPATH 22
8.17 ERRORPATH 23
8.18 JOINFILES 23
8.19 TRANSFERFILES 24
8.20 DEADLINETIME 24
8.21 HARDWALLCLOCKTIMELIMIT 24
8.22 SOFTWALLCLOCKTIMELIMIT 24
8.23 HARDRUNDURATIONLIMIT 24
8.24 SOFTRUNDURATIONLIMIT 24
8.25 ATTRIBUTENAMES 25

9 SESSION INTERFACE 25

9.1 CONSTANTS 25
9.2 INIT 26
9.3 EXIT 27
9.4 CREATEJOBTEMPLATE 27
9.5 DELETEJOBTEMPLATE 28
9.6 RUNJOB 28
9.7 RUNBULKJOBS 29
9.8 CONTROL 30
9.9 SYNCHRONIZE 32
9.10 WAIT 33
9.11 JOBPROGRAMSTATUS 34
9.12 CONTACT 35
9.13 VERSION 36
9.14 DRMSINFO 36
9.15 DRMAAIMPLEMENTATION 36

10 ANNEX 37

10.1 COMPLETE IDL SPECIFICATION 37
10.2 CORRELATION OF DRMAA EXCEPTIONS AND ERROR CODES 41
10.3 CORRELATION OF *JOBTEMPLATE* ATTRIBUTES AND ATTRIBUTE NAME STRINGS 42

11 SECURITY CONSIDERATIONS 43

12 REFERENCES 43

13 CONTRIBUTORS 43

14 ACKNOWLEDGEMENTS 44
15 INTELLECTUAL PROPERTY STATEMENT 44
16 DISCLAIMER 44
17 FULL COPYRIGHT NOTICE 45

1 Introduction

This document gives an IDL description for the DRMAA interface. It arises from the results of a collaborative effort to bring the Java™ language binding and .NET language binding into agreement, based on the DRMAA 1.0 specification.
The DRMAA 1.0 specification was written originally with a procedural C-language slant. As such, several aspects of the DRMAA interface needed to be altered slightly to better fit with object-oriented languages. Among the aspects that changed are variable and method naming and the error structure.
Although this document can be seen as stand-alone, it still bases on the concepts defined in the DRMAA 1.0 specification. The text refers to the respective chapter of the DRMAA standard whenever it is necessary.

1.1 Notational Conventions

In this document, the following conventions are used:

- `IDL language elements and definitions` are represented in a fixed-width font.
- *References to IDL language elements and definitions* are represented in italics.

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

The document describes the DRMAA interface semantics with the help of OMG IDL [OMG IDL]. It includes a set of overall rules for the creation of specific language bindings for the given specification. Specific examples are given for the Java language. These examples are not normative.

2 General Concepts

2.1 Design Decisions

An effort has been made to choose design patterns that are not unique to a specific language. However, in some cases, various languages disagree over some points. In those cases, the most meritous approach was taken, irrespective of language.
The following text bases on the terminology of OMG IDL. For this reason, all operational semantics are described in terms of interfaces and not of classes. This concept ensures the possibility to map the described operational semantics to a variety of object-oriented, and even procedural, languages. The usage of a class concept depends on the specific language-mapping rules. The IDL specification assumes that destination languages for a binding typically support the concepts of *exceptions*.
If a destination language does not support the notion of exceptions (like ANSI C), the language binding SHOULD map error conditions to an appropriate consistent concept. A language binding MAY chose to model exceptions as numeric error code return values, and return values as additional output parameters of the operation.

2.2 IDL language mapping

Language binding documents based on this specification MUST define a mapping between the IDL constructs used in this specification and their specific language constructs. A language binding SHOULD NOT rely itself completely on the OMG language mapping documents available for many programming languages. It must be considered that the OMG mappings bring a huge overhead of irrelevant CORBA-related mapping rules into the specification. Therefore it must be carefully decided whether a binding decision reflects a natural and simple mapping of the intended purpose for the DRMAA interfaces. In most situations it SHOULD be enough to reuse value type mappings only and to define custom mappings for the reference types.

The language binding MUST use the described concept mapping in a consistent manner for the overall specification.

It may be the case that IDL constructs do not map directly to an according language construct. In this case it MUST be ensured that the according construct in the particular language retains the intended semantic of the DRMAA interface definition.

Languages without an explicit notion of enumerations MAY map the IDL enumeration values to constant class members, enabled by the distinct naming of all enumeration values in the specification.

Some attributes and operation parameters are scoped ("DRMAA::"), in order to avoid naming clashes in case-insensitive programming languages. Language bindings for case-sensitive languages SHOULD omit this explicit scoping.

This specification tries to consider the possibility of a Remote Procedure Call scenario in a DRMAA-conformant language mapping. It SHOULD therefore be ensured that the programming language type for an IDL *valuetype* definition supports the serialization and comparison of *valuetype* instances. These capabilities SHOULD be accomplished through whatever mechanism is most natural for the specific programming language.

Java binding example:

IDL	*Java language*
`module` definition	`package` keyword
`interface` definition	`public abstract interface` definition
`enum` definition with enumeration members	Enumeration members become Java `int` constants in the surrounding interface definition
`string` type	`java.lang.String`
`long` type	`int`
`long long` type	`long`
`const` type	`public static final`
`boolean` type	`boolean`
`[readonly] attribute` type	Getter [and setter] methods in JavaBeans™ style, boolean readonly attribute names are

	prefixed with “get”.
`exception` type	Class definition, derived from `java.lang.Exception`
`raises` clause	`throws` clause
`valuetype` definition	`public class` definition, may additionally implement the *Cloneable*, *Serializable*, and *Compareable* interfaces
`factory` definition	class constructor

The DRMAA IDL definition defines specialized custom types as new value types, in order to express their intended semantics:

```
// unbounded native ordered string list
valuetype OrderedStringList sequence<string>;
// unbounded native string list
valuetype StringList sequence<string>;
// dictionary type, for unbounded key-value pair storage
valuetype Dictionary sequence< sequence<string,2> >;
// amount of time, at least with a resolution to seconds
valuetype TimeAmount long long;
```

The language-binding author SHOULD replace these type definitions directly with semantically equal references or value types from the according language. This MAY include the creation of new complex language types for one or more of the above concepts, depending on the context.

Java binding example:

IDL	*Java*
`StringList`	`java.util.Set`
`OrderedStringList`	`java.util.List`
`TimeAmount`	`long`
`Dictionary`	`java.util.Map`

3 The DRMAA API Module

The DRMAA interfaces and structures are encapsulated by a naming scope, which avoids conflicts with other API’s used in the same application.

```
module DRMAA{
   …
}
```

Language binding authors MUST map the IDL module encapsulation to an according package or namespace concept and MAY change the module name according to programming language conventions.

Java binding example:

IDL	*Java*
`module DRMAA`	`package org.drmaa`

4 Data Types

4.1 JobControlAction enumeration

The *JobControlAction* enumeration is used as a input parameter type by the *control()* method in the *Session* interface. The meanings of the enumeration values are specified in the description of the method in section 9.8.

```
enum JobControlAction {
   SUSPEND,
   RESUME,
   HOLD,
   RELEASE,
   TERMINATE
};
```

4.2 JobProgramState enumeration

The *JobProgramState* enumeration is used as a input parameter type by the *jobProgramStatus()* method in the *Session* interface. The meanings of the enumeration values are specified in the description of the method in section 9.11. A DRMAA implementation is not required to be able to return all of the job state values in the JobProgramState enumeration. If a given job state has no representation in the underlying DRMS, the DRMAA implementation MAY ignore that job state value. All DRMAA implementations MUST, however, define the JobProgramState enumeration, and the definition MUST include **all** job state values, including those for unused job states. An implementation SHOULD NOT return any job state value other than those defined in the JobProgramState enumeration.

```
enum JobProgramState {
   UNDETERMINED,
   QUEUED_ACTIVE,
   SYSTEM_ON_HOLD,
   USER_ON_HOLD,
   USER_SYSTEM_ON_HOLD,
   RUNNING,
   SYSTEM_SUSPENDED,
   USER_SUSPENDED,
   USER_SYSTEM_SUSPENDED,
   DONE,
   FAILED
};
```

The status values relate to the DRMAA job state transition model, as shown in Figure 1.

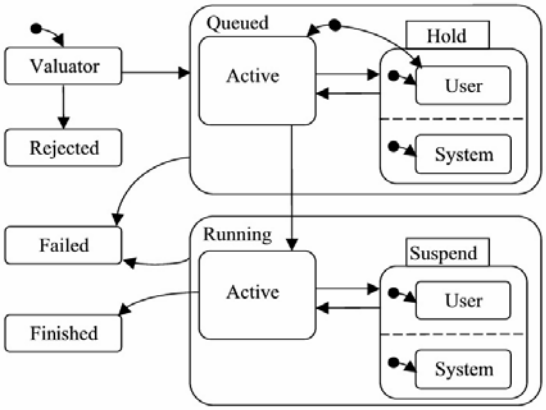

Figure 1: DRMAA Job State Transition Diagram

4.3 JobSubmissionState enumeration

The *JobSubmissionState* enumeration is used as the type of the *JobTemplate::jobSubmissionState* interface attribute. In the context of the job template, the enumeration values have the following meaning:

- *HOLD_STATE:* The job may be queued, but it is not eligible to run.
- *ACTIVE_STATE:* The job is eligible to run.

```
enum JobSubmissionState {
          HOLD_STATE,
          ACTIVE_STATE
};
```

4.4 FileTransferMode value type

The *FileTransferMode* value-type is used by a *JobTemplate* instance to indicate the value for the *transferFiles* attribute. The type contains three attributes which determine the streams that will be staged in or out.

```
valuetype FileTransferMode {
   attribute boolean transferInputStream;
   attribute boolean transferOutputStream;
   attribute boolean transferErrorStream;
};
```

4.4.1 transferInputStream

This attribute defines whether to transfer an input stream file. If this attribute contains true, the transfer*inputStream* attribute of the corresponding job template SHALL be treated as the source from which the input file should be copied.

4.4.2 transferOutputStream

This attribute defines whether to transfer an output stream file. If this attribute contains true, the transferO*utputStream* attribute of the corresponding job template SHALL be treated as the destination to which the output file should be copied.

4.4.3 transferErrorStream

This attribute defines whether to transfer an error stream file. If this attribute contains true, the transferE*rrorStream* attribute of the corresponding job template SHALL be treated as the destination to which the error file should be copied.

4.5 Version value type

The *Version* value type is a holding structure for the major and minor version numbers of the DRMAA implementation as contained in the *version* attribute of the *Session* interface. The string representation (see section 2.2) of a *Version* instance MUST be of the form "<major>.<minor>".

```
valuetype Version {
   readonly attribute long major;
   readonly attribute long minor;
};
```

4.5.1 major

This attribute SHALL contain the major version number.

4.5.2 minor

This attribute SHALL contain the minor version number.

5 Exceptions

All exceptions in specific bindings MUST contain a possibility to store and read a textual description of the exception cause for the exception instance.
Language bindings MAY decide to derive all exceptions from given environmental exception base class(es). Language bindings SHOULD replace exceptions with a semantically equivalent native runtime environment exception whenever this is appropriate.

```
exception AlreadyActiveSessionException {string message;};
exception AuthorizationException {string message;};
exception ConflictingAttributeValuesException {string message;};
exception DefaultContactStringException {string message;};
exception DeniedByDrmException {string message;};
exception DrmCommunicationException {string message;};
exception DrmsExitException {string message;};
exception DrmsInitException {string message;};
exception ExitTimeoutException {string message;};
exception HoldInconsistentStateException {string message;};
exception IllegalStateException {string message;};
exception InternalException {string message;};
exception InvalidArgumentException {string message;};
exception InvalidAttributeFormatException {string message;};
exception InvalidAttributeValueException {string message;};
exception InvalidContactStringException {string message;};
```

```
exception InvalidJobException {string message;};
exception InvalidJobTemplateException {string message;};
exception NoActiveSessionException {string message;};
exception NoDefaultContactStringSelectedException {string message;};
exception OutOfMemoryException {string message;};
exception ReleaseInconsistentStateException {string message;};
exception ResumeInconsistentStateException {string message;};
exception SuspendInconsistentStateException {string message;};
exception TryLaterException {string message;};
exception UnsupportedAttributeException {string message;};
```

Language bindings MAY decide to introduce a hierarchical ordering of the DRMAA exceptions through class derivation. In this case it MAY also happen that new exceptions are introduced for behavior aggregation. In this case, those exceptions SHALL be marked as abstract, to prevent them from being thrown.

If the language supports the distinction between static ('checked') and runtime ('unchecked') exceptions, all but the following exceptions must be represented as checked exception:

- `InternalException`
- `OutOfMemoryException`
- `InvalidArgumentException`

If a destination language does not support the notion of exceptions (like ANSI C), the language binding SHOULD map error conditions to an appropriate consistent concept. A language binding MAY chose to model exceptions as numeric error code return values, and return values as additional output parameter of the operation. The mapping of exceptions to error codes is presented in Section 10.2. A language binding SHOULD specify numeric values for all DRMAA error constants.

5.1 AlreadyActiveSessionException

Initialization failed due to existing DRMAA session.

5.2 AuthorizationException

The user is not authorized to perform the given operation.

5.3 ConflictingAttributeValuesException

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

5.4 DefaultContactStringException

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

5.5 DeniedByDrmException

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

5.6 DrmCommunicationException

Could not contact DRM system.

5.7 DrmsExitException

A problem was encountered while trying to exit the session.

5.8 DrmsInitException

A problem was encountered while trying to initialize the session.

5.9 ExitTimeoutException

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

5.10 HoldInconsistentStateException

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

5.11 InternalException

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

5.12 InvalidArgumentException

A parameter value is fundamentally invalid, such as being of the wrong type or being `null`.

5.13 InvalidAttributeFormatException

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

5.14 InvalidAttributeValueException

The value for the job template property is invalid.

5.15 InvalidContactStringException

The given contact string is not valid.

5.16 InvalidJobException

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

5.17 InvalidJobTemplateException

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

5.18 NoActiveSessionException

Method call failed because there is no active session.

5.19 NoDefaultContactStringSelectedException

No defaults contact string was provided or selected. DRMAA requires that the default contact string is selected when there is more than one default contact string due to multiple DRMAA implementations being present and available (see also 9.2).

5.20 OutOfMemoryException

This exception can be thrown by any method at any time when the DRMAA implementation has run out of free memory.

5.21 ReleaseInconsistentStateException

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

5.22 ResumeInconsistentStateException

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

5.23 SuspendInconsistentStateException

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

5.24 TryLaterException

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

5.25 UnsupportedAttributeException

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

5.26 IllegalStateException

The JobInfo instance is not in the correct state for this kind of operation.

6 The PartialTimestamp

The *PartialTimestamp* type is used by *JobTemplate* interface instances to represent partially specified time stamps, as required by the Distributed Resource Management Application API Specification 1.0. The *PartialTimestamp* SHOULD be an extension of the native language date/time representation if possible and reasonable. For this reason, the following text describes the functional requirements without a specific signature for the type definition. The IDL definition covers this aspect by specifying a native data type.

```
native PartialTimestamp;
```

The *PartialTimestamp* MUST support the following fields: century (>=19), year (0-99), month (1-12), date (1-31), hour (0-23), minute (0-59), second (0-61), zone offset hour (-11 - 12), and zone offset minute (0-59). It MUST support the following essential operations: “get field value”, “set field value”, “get time as native date/time object”, “convert to string” and “parse from string.” If possible, these operations SHOULD leverage structure already present in the native date/time class, even if this leads to a mapping with multiple classes or interfaces. The two field operations MAY be represented as attributes.
The “get field value” operation MUST return the current value for the given field. The “set field value” operation MUST set the current value for the given field. The “get time as native date/time object” operation MUST resolve the partial time to a specific time that is the soonest possible time that is not in the past, and SHOULD return that specific time as a native date/time representation. The “convert to string” operation MUST return the partial time represented by the *PartialTimestamp* as a string which adheres to the following format: `[[[[CC]YY/]MM/]DD] hh:mm[:ss] [{-|+}UU:uu]`, where:

- CC is the first two digits of the year [19,]
- YY is the last two digits of the year [0,99]
- MM is the two digits of the month [01,12]
- DD is the two-digit day of the month [01,31]
- hh is the two-digit hour of the day [00,23]
- mm is the two-digit minute of the day [00,59]
- ss is the two-digit second of the minute [00,61]
- UU is the two-digit hours since (before) UTC [-11,12]
- uu is the two-digit minutes since (before) UTC [0,59]

In order for this operation to be performed, the *PartialTimestamp* must have no unset field of a lower order than the highest order set field, with the exception of the second and zone offset fields. For example, if the year is set, the month, date, hour, and minute must also be set for this operation to be performed. Failure to meet this criterion MUST result in an *InvalidArgumentException* being thrown, or the corresponding error code being returned in languages which do not support exceptions. The seconds and UTC offset are always optional. The “parse from string” operation MUST parse a string in the above format to generate a *PartialTimestamp* as the return value. If the string is not in the above format, an *InvalidArgumentException* or an appropriate language-dependent exception MUST be thrown or the corresponding error code MUST be returned in languages that do not support exceptions. If a PartialTimestamp type is resolved to a concrete time before all fields are set, the unset fields SHALL be filled in using the current time in such a way that the resulting concrete time is the soonest possible time which agrees with the set fields and is not in the past. A

PartialTimestamp type MAY be resolved to a concrete time any number of times. Each resolution will result in a concrete time that meets the above criteria for the point in time at which the resolution took place.
The resolving of partial time information MUST be performed according to the following rules:

- If the optional UTC-offset is not specified, the offset associated with the local timezone SHALL be used.
- If the second is not specified, then it SHALL be treated as zero.
- If the day is not specified, the current day SHALL be used unless the specified hour, minute and second has already elapsed, in which case the next day SHALL be used.
- If the month is not specified, the current month SHALL be used unless the specified day, hour, minute and second has already elapsed, in which case the next month SHALL be used.
- If the year is not specified, the current year SHALL be used unless the specified month, day, hour, minute and second has already elapsed, in which case the next year SHALL be used.
- If the century is not specified, the current century SHALL be used unless the specified year, month, day, hour, minute and second has already elapsed, in which case the next century SHALL be used.

The *PartialTimestamp* MAY also support the following four operations: “get field modifier,” “set field modifier,” “add to field,” and “roll field.” If possible, these operations SHOULD leverage structure already present in the native language date/time representation. The “get field modifier” operation MUST return any additional modifiers set for the given field. An additional modifier is added to the field's value after it has been resolved to a specific time. The “set field modifier” operation MUST set the additional modifiers for the given field. The “add to field” operation MUST add a given value to the given field. If supported by the native date/time representation, this operation SHOULD attempt to resolve out of range field values that may result from the operation. For example, adding “1” to the date of a *PartialTimestamp* instance which is set to January 31st SHOULD result in the *PartialTimestamp* being set to February 1st. If this operation is supported, the “get field modifier” and “set field modifier” operations MUST also be supported. The “roll field” operation is similar to the “add to field” operation, except that the operation cannot modify a field of a higher order than the given field. Such modifications are simply lost. For example, adding “1” to the date of a *PartialTimestamp* which is set to January 31st SHOULD result in the *PartialTimestamp* being set to January 1st.
The *PartialTimestamp* MUST also support a notion of unset fields. A special value is assigned to all fields which have not been explicitly set. This special value MUST be of the same type as the date/time properties and MAY be the maximum value for that data type.
Language bindings are free to define convenience functions in addition to the functionalities described here.

7 JobInfo interface

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

```
interface JobInfo {
   readonly attribute string jobId;
   readonly attribute Dictionary resourceUsage;
   readonly attribute boolean hasExited;
   readonly attribute long exitStatus;
   readonly attribute boolean hasSignaled;
```

```
    readonly attribute string terminatingSignal;
    readonly attribute boolean hasCoreDump;
    readonly attribute boolean wasAborted;

};
```

In languages which do not support the notion of interfaces and objects, the JobInfo interface SHOULD be modeled as a series of routines which utilize an opaque job object returned from the wait() routine.

The following sections explain the meanings of the *JobInfo* member attributes.

7.1 jobId

The identifier for the completed job.

7.2 resourceUsage

This attribute SHALL contain the completed job's resource usage data. If the job did not produce resource usage data, this attribute SHALL be null. Please refer also to [DRMAA10] section 3.1.3 for more information about resource usage data semantics.

7.3 hasExited

This attribute SHALL contain *true* if the job terminated normally. A value of *false* MAY indicate that although the job has terminated normally, an exit status is not available, or that it is not known whether the job terminated normally. In both cases the *exitStatus* attribute SHALL NOT contain exit status information. A value of *true* indicates more detailed diagnosis can be retrieved from the *exitStatus* attribute.

7.4 exitStatus

If *exited* is *true*, this attribute SHALL contain the operating system exit code of the job. If *exited* is *false*, the getter function for this attribute MUST raise an *IllegalStateException*.

7.5 hasSignaled

This attribute SHALL contain *true* if the job terminated due to the receipt of a signal. A value of *false* MAY also indicate that although the job has terminated due to the receipt of a signal, the signal is not available, or that it is not known whether the job terminated due to the receipt of a signal. In both cases *terminatingSignal* SHALL NOT provide signal information.

7.6 terminatingSignal

If *hasSignaled* is *true*, this attribute SHALL contain 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.
If *hasSignaled* is *false*, the getter function for this attribute MUST raise an *IllegalStateException*.

7.7 hasCoreDump

If *hasSignaled* is *true*, this attribute SHALL contain *true* if a core image of the terminated job was created.
If *hasSignaled* is *false*, the getter function for this attribute MUST raise an *IllegalStateException*.

7.8 wasAborted

This attribute SHALL contain *true* if the job ended before entering the running state.

8 JobTemplate interface

In order to define the attributes associated with a job, a DRMAA application uses the *JobTemplate* interface. Instances of such templates are created via the active *Session* implementation. A DRMAA application gets a *JobTemplate* from the active *Session* instance, specifies in the template any required job parameters, and the passes the template back to the DRMAA Session instance when requesting that a job be executed. When finished, the DRMAA application SHOULD call the *Session::deleteJobTemplate()* method to allow the underlying implementation to free any resources bound to the *JobTemplate* instance. Please refer also to [DRMAA10] section 3.1.4 to 3.1.6 for more information regarding precedence rules, site-specific requirements and job evaluation.

A language binding specification MUST model the *JobTemplate* interface in the following way:

```
interface JobTemplate{
   const string HOME_DIRECTORY = "$drmaa_hd_ph$";
   const string WORKING_DIRECTORY = "$drmaa_wd_ph$";
   const string PARAMETRIC_INDEX = "$drmaa_incr_ph$";
   attribute string remoteCommand;
   attribute OrderedStringList args;
   attribute DRMAA::JobSubmissionState jobSubmissionState;
   attribute Dictionary jobEnvironment;
   attribute string workingDirectory;
   attribute string jobCategory;
   attribute string nativeSpecification;
   attribute StringList email;
   attribute boolean blockEmail;
   attribute PartialTimestamp startTime;
   attribute string jobName;
   attribute string inputPath;
   attribute string outputPath;
   attribute string errorPath;
   attribute boolean joinFiles;
   attribute FileTransferMode transferFiles;
   attribute PartialTimestamp deadlineTime;
   attribute TimeAmount hardWallclockTimeLimit;
   attribute TimeAmount softWallClockTimeLimit;
   attribute TimeAmount hardRunDurationLimit;
   attribute TimeAmount softRunDurationLimit;
   readonly attribute StringList attributeNames;
      …
   [language-specific operations for implementation-specific attributes]
      …
```

In languages which do not support the notion of interfaces or objects, the job template attributes SHOULD be modeled as constant parameters to generic getter and setter routines. These routines SHOULD treat all attribute names and values as strings. In the case of such a language, the attributeNames attribute SHOULD be modeled as a getAttributeNames() routine that returns the names of the available attributes as a list of string which can be used with the generic getter and setter routines. See section 8.1.1 below.

The *JobTemplate* implementation MUST support the following exceptions for the setter operations in case there is a concept of exceptions in the programming language:

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

For both getter and setter operations, the following exceptions MUST be supported in case exceptions are part of the programming language:

- *NoActiveSessionException*
- *DrmCommunicationException*
- *AuthorizationException*
- *OutOfMemoryException*
- *InternalException*

In most cases, a DRMAA implementation will require that job templates be created through the *Session::createJobTemplate()* method. In those cases, passing a template created other than via this method to the *Session::deleteJobTemplate()*, *Session::runJob()*, or *Session::runBulkJobs()* methods MUST result in an *InvalidJobTemplateException* being thrown or a corresponding error code being returned if exceptions are not supported.

A *JobTemplate* instance SHOULD be convertible to a string for printing. This SHOULD be accomplished through whatever mechanism is most natural for the implementation language. The resulting string MUST contain the values of all set properties.

Access to scalar attributes (`string, Boolean, long`) MUST operate in a pass-by-value mode. An according language binding must ensure that this behavior is always fulfilled. For non-scalar attributes, the language binding MUST specify a consistent access strategy for all these attributes – either pass-by-value or pass-by-reference – according to the use cases of language binding implementations.

In the DRMAA job template concept, there is a distinction between mandatory, optional and implementation-specific attributes. A language binding implementation MUST include all DRMAA attributes described here, both required and optional. The setter and getter implementations for optional attributes MUST in case throw *UnsupportedAttributeException*. The service provider implementation SHOULD then override the setters and getters for supported optional attributes with methods that operate normally. In the case of a destination language which does not support the notion of interfaces or objects, the generic getter and setter routines should throw *UnsupportedAttributeException* when called with the name of an unknown or unsupported attribute.

8.1.1 Generic getter / setter routines

In the case of a destination language which does not support the notion of interfaces or objects, the JobTemplate interface SHOULD be modeled by a set of generic setter and getter routines. These generic routines are as follows:

```
string getAttribute(string name)
```

```
   raises ( DrmCommunicationException,
     AuthorizationException,
     NoActiveSessionException,
     OutOfMemoryException,
     InternalException,
     UnsupportedAttributeException);
};
```

This method SHALL return the string value of the specified attribute. The language binding specification SHOULD consistently specify the string representation for non-string data types. Valid input values are the strings returned by the *getAttributeNames()* operation. An invalid attribute name leads to an *UnsupportedAttributeException*.

```
stringlist getVectorAttribute(string name)
   raises ( DrmCommunicationException,
     AuthorizationException,
     NoActiveSessionException,
     OutOfMemoryException,
     InternalException,
     UnsupportedAttributeException);
};
```

This method SHALL return the list of string values of the specified vector attribute. A vector attribute is one which is prefixed with "v_" in the table in section 10.3. The language binding specification SHOULD consistently specify the string representation for non-string vector elements. Valid input values are the strings returned by the *getAttributeNames()* operation. An invalid attribute name leads to an *UnsupportedAttributeException*.

```
void setAttribute(string name, string value)
   raises ( DrmCommunicationException,
     UnsupportedAttributeException,
     InvalidAttributeValueException,
     AuthorizationException,
     NoActiveSessionException,
     OutOfMemoryException,
     InternalException);
};
```

This method SHALL change the value of the specified attribute to the given value. Valid input values for the *name* parameter are the strings returned by the *getAttributeNames()* operation. An invalid attribute name leads to an *UnsupportedAttributeException*. An invalid value for a particular attribute leads to an *InvalidAttributeValueException*. The language binding specification SHOULD consistently specify the string representation for non-string data types.

```
void setVectorAttribute(string name, stringlist value)
   raises ( DrmCommunicationException,
     UnsupportedAttributeException,
     InvalidAttributeValueException,
     AuthorizationException,
     NoActiveSessionException,
     OutOfMemoryException,
     InternalException);
};
```

This method SHALL replace the list of values of the specified vector attribute to the given list of values. A vector attribute is one which is prefixed with "v_" in the table in section 10.3. Valid input values for the *name* parameter are the strings returned by the *getAttributeNames()* operation. An invalid attribute name leads to an *UnsupportedAttributeException*. An invalid

value for a particular attribute leads to an *InvalidAttributeValueException*. The language binding specification SHOULD consistently specify the string representation for non-string vector elements.

If a language binding uses this generic getter / setter approach, then it MUST enforce the usage of the attribute names specification from Section 10.3. for all implementations, and all attributes listed in section 10.3 MUST be implemented.

8.2 Accessing Implementation-specific Attributes

A language binding MUST provide a means for accessing implementation-specific attributes, as the getters and setters for such attributes are not defined by the *JobTemplate* interface. This access method MUST be consistent for all attributes and SHOULD be clearly described in the language binding specification. Some destination languages MAY enable more than one access mechanism.

Some common approaches are:

8.2.1 Introspection approach

In order to access the getters and setters for implementation-specific attributes, the developer must use the destination language's introspection mechanisms to locate and then call the attributes' getters and setters at run time. In such a case, the list of attribute names given by the *attributeNames* attribute MUST be names which are meaningful to the destination language's introspection mechanism.

This approach makes it possible to write applications which are completely portable across binding implementations, including previously unknown binding implementations assuming that the naming of implementation-specific attributes is consistent and/or predictable. A significant disadvantage to this approach is the complexity or writing fully dynamic, introspection-based application logic.

8.2.2 Dynamic Loader Approach

In languages which support dynamic class loading, access to implementation-specific attributes can be encapsulated in classes dedicated to accessing the job template attributes of a specific binding implementation. After determining the binding implementation in use, an application in such a language could dynamically load a class which is capable of setting the implementation-specific attributes of the job template.

An advantage of this approach is that within the scope of the dynamically loaded class, the job template may be safely cast to the implementation type without creating a run-time dependency on the implementation class. Within the class access to the job template attributes is done directly using the job template implementation's declared getters and setters. A disadvantage is that such a class is needed for each binding implementation to be supported, and each such class is limited to operating only on that specific binding implementation. Another disadvantage is that it creates a compile-time dependency on all supported binding implementations, i.e. **all** supported binding implementations must be available at the time the application is compiled.

8.2.3 Discouraged approaches

The direct casting of a job template to the job template implementation class without the use of dynamic class loading SHOULD NOT be used. Such casting, while enabling direct access to all job template attribute getters and setters, creates a compile-time and run-time dependency on all supported binding implementations, i.e. such an application must be bundled with **all** binding implementations, even if it will only be run on one of them.
Also the combination of job template attribute getters and setters with generic getters and setters, where either set of accessors provides access to only a subset of the job template

implementations attributes, SHOULD NOT be used. A DRMAA binding MUST provide consistent attribute access, with support for all attribute types (required, optional and implementation-specific) in only one language-specific method.

8.3 Constants

The *JobTemplate* interface defines a set of constants which are used in the context of some of the attributes:

```
const string HOME_DIRECTORY = "$drmaa_hd_ph$";
const string WORKING_DIRECTORY = "$drmaa_wd_ph$";
const string PARAMETRIC_INDEX = "$drmaa_incr_ph$";
```

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* attributes.

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

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* attributes.

8.4 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 attribute value SHOULD NOT relate to binary file management or file staging activities.

8.5 args

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

8.6 jobSubmissionState

Defines the state of the job at submission time. For more information see section 4.3.

8.7 jobEnvironment

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

8.8 workingDirectory

This attribute specifies the directory where the job is executed. If the attribute is not set, the behavior is is implementation dependent. The attribute value MUST be evaluated relative to the execution host's file system. The attribute value MAY contain the *HOME_DIRECTORY* or *PARAMETRIC_INDEX* constant values as placeholders. A *HOME_DIRECTORY* placeholder at

the begin 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 attribute value in the case of parametric job templates and SHALL be substituted by the underlying DRM system with the parametric jobs' index.
The *workingDirectory* MUST be specified in a syntax that is common at the host where the job is executed.
If the attribute is set and no placeholder is used, an absolute directory specification is expected.
If the attribute is set and the job was submitted successfully and the directory does not exist, the job MUST enter the state, `JobProgramState.FAILED`.

8.9 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 attribute. 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.

8.10 nativeSpecification

An implementation-defined string that is passed by the end user to DRMAA to specify site-specific resources and/or policies.
As far as the DRMAA interface specification is concerned, the native specification is an implementation-defined string and is interpreted by each DRMAA library. One MAY use the job category and the native specification with the same job submission for policy specification. In this case, the DRMAA library is assumed to be capable of merging the outcome of the two policy sources in a reasonable way.
The native specification MAY be used without the requirement to maintain job categories, and submit options MAY be specified directly.
More information can be found in section 2.4.2 of the DRMAA 1.0 specification document.

8.11 email

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

8.12 blockEmail

This Boolean parameter decides whether the sending of email is blocked by default or not, regardless of the DRMS setting. If the parameter is TRUE, the sending of email SHALL be blocked regardless of the DRMS setting. If the value is FALSE, the sending of email SHALL be determined by the DRMS setting.

8.13 startTime

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

8.14 jobName

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

8.15 inputPath

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

```
[hostname]:file_path
```

If the *transferFiles* job template attribute is supported and has a value where the *FileTransferMode::inputStream* attribute set to *true*, the input file SHOULD be fetched by the underlying DRM system from the specified host, or from the submit host if no hostname was specified.
If the *transferFiles* job template attribute is not supported or its value's *FileTransferMode::inputStream* is set to *false*, then the input file is always expected at the host where the job is executed, irrespective of whether a hostname was specified.
The *PARAMETRIC_INDEX* placeholder can be used at any position for parametric job templates and SHALL be substituted by the underlying DRM system with the parametric job's index.
A *HOME_DIRECTORY* placeholder at the beginning of the attribute value denotes the remaining portion as a relative file specification resolved relative to the job's user's home directory at the host where the file is located.
A *WORKING_DIRECTORY* placeholder at the beginning of the attribute value denotes the remaining portion as a relative file specification resolved relative to the job's working directory at the host where the file is located.
The *inputPath* MUST be specified in a syntax that is common at the host where the file is located.
If set, and the job was successfully submitted, and the file can't be read, the job enters the state, *JobProgramState.FAILED*.

8.16 outputPath

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

```
[hostname]:file_path
```

If the *transferFiles* job template attribute is supported and its value's *FileTransferMode::outputStream* attribute is set to *true*, the output file SHALL be transferred by the underlying DRM system to the specified host or to the submit host if no hostname is specified.
If the *transferFiles* job template attribute is not supported or its value's *FileTransferMode::outputStream* attribute is set to *false,* the output file SHALL be kept at the host where the job is executed, irrespective of whether a hostname was specified.
All output sent to the job's standard output stream SHALL be appended to that file. If the file does not exist at the time the job is executed, the file SHALL first be created.

The *PARAMETRIC_INDEX* placeholder can be used at any position with parametric job templates and SHALL be substituted by the underlying DRM system with the parametric job's index.
A *HOME_DIRECTORY* placeholder at the beginning denotes the remaining portion as a relative file specification resolved relative to the job users home directory at the host where the file is located.
A *WORKING_DIRECTORY* placeholder at the beginning denotes the remaining portion as a relative file specification resolved relative to the jobs working directory at the host where the file is located.
The *outputPath* MUST be specified in a syntax that is common at the host where the file is located. If set and the job was successfully submitted and the file can't be written before execution the job MUST enter the state, `JobProgramState.FAILED`.

8.17 errorPath

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

```
[hostname]:file_path
```

If the *transferFiles* job template attribute is supported and it's value's *FileTransferMode::errorStream* attribute is set to *true*, the error file SHALL be transferred by the underlying DRM system to the specified host or to the submit host if no hostname is specified.
If the *transferFiles* job template attribute is not supported or it's value's *FileTransferMode::errorStream is* set to *false*, the error file is always kept at the host where the job is executed irrespective of whether a hostname was specified.
All output sent to the job's standard error stream SHALL be appended to that file. If the file does not exist at the time the job is executed, the file SHALL first be created.
The *PARAMETRIC_INDEX* placeholder can be used at any position for parametric job templates and SHALL be substituted by the underlying DRM system with the parametric jobs' index.
A *HOME_DIRECTORY* placeholder at the beginning denotes the remaining portion as a relative file specification, resolved relative to the job users home directory at the host where the file is located.
A *WORKING_DIRECTORY* placeholder at the beginning denotes the remaining portion as a relative file specification resolved relative to the jobs working directory at the host where the file is located.
The *errorPath* MUST be specified in a syntax that is common at the host where the file is located.
If set and the job was successfully submitted and the file can't be written before execution, the job enters the state, `JobProgramState.FAILED`.

8.18 joinFiles

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

8.19 transferFiles

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

This attribute is optional. An implementation MUST throw an *UnsupportedAttributeException* if this attribute is not supported.

8.20 deadlineTime

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

This attribute is optional. An implementation MUST throw an *UnsupportedAttributeException* if this attribute is not supported.

8.21 hardWallclockTimeLimit

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

This attribute is optional. In case an implementation MUST throw an *UnsupportedAttributeException* if this attribute is not supported.

8.22 softWallClockTimeLimit

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

This attribute is optional. In case an implementation MUST throw an *UnsupportedAttributeException* if this attribute is not supported.

8.23 hardRunDurationLimit

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

This attribute is optional. In case an implementation MUST throw an *UnsupportedAttributeException* if this attribute is not supported.

8.24 softRunDurationLimit

This attribute specifies an estimate as to how long the job will need to remain in a running state to complete. This attribute is intended to assist the scheduler. If the time specified is insufficient, the implementation MAY impose a scheduling penalty.

This attribute is optional. In case an implementation MUST throw an *UnsupportedAttributeException* if this attribute is not supported.

8.25 attributeNames

This read-only attribute specifies the list of supported attribute names. This list includes supported DRMAA reserved attribute names (both required and optional) and implementation-specific attribute names. The listed attribute name MUST be of a format that is meaningful to the destination language for use in introspection, if supported, or with the getAttribute() and setAttribute() methods if introspection is not supported. See 10.3 for a given names of the job template attributes.

9 Session interface

The following chapter explains the set of constants, methods and attributes defined in the Session interface. Please consult [DRMAA10] section 3.1.2 for further details about the DRMAA session concept.

```
interface Session{
```

9.1 Constants

The Session interface defines a set of constant values, which are used in the context of several interface functions.

```
const long long TIMEOUT_WAIT_FOREVER = -1;
const long long TIMEOUT_NO_WAIT = 0;
const string JOB_IDS_SESSION_ANY = "DRMAA_JOB_IDS_SESSION_ANY";
const string JOB_IDS_SESSION_ALL = "DRMAA_JOB_IDS_SESSION_ALL";
```

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

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

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

The *JOB_IDS_SESSION_ALL* constant is used with the *control()* and *synchronize()* methods to indicate that a method call should operate on all jobs in the session at submission time, minus any jobs that go out of scope during the run time of the operation. For example: If a job was in the session at the time of calling *synchronize(JOB_IDS_SESSION_ALL)*, and it's gets reaped during the operation, the overall call will not fail. A call with JOB_IDS_SESSION_ALL to an empty session SHALL result in a successful call. In case that a call with *JOB_IDS_SESSION_ALL* fails for a partial set of the jobs in the session, the implementation SHALL throw an *InternalException*. The error text of the exception should explain the problem in detail and may give an idea of the current status of the session.

9.2 init

The *init()* method MUST do whatever work is required to initialize a DRMAA session for use. The *contactString* parameter is an implementation-dependent string that may be used to specify which DRM system to use. This method must be called before any other DRMAA calls, except for the getter functions of the *contact*, *drmsInfo*, and *drmaaImplementation* attributes defined in the Session interface.
If contact is `null` or emtpy, the default DRM system SHOULD be used, provided there is only one DRMS available. If contact is `null` or empty, and more than one DRMAA implementation is available, *init()* SHALL throw a *NoDefaultContactStringSelectedException* or return a corresponding error code if exceptions aren't supported.
init() SHOULD be called only once, by only one of the threads. The main thread is recommended. A call to *init()* by another thread or additional calls to *init()* by the same thread SHOULD throw an *AlreadyActiveSessionException* or return a corresponding error code if exceptions are not supported.
In the case that a DRMAA library implementation needs to perform non-thread-safe operations (like getHostByName() C library call), it SHOULD perform them in the implementation of the init() operation, in order to ensure thread-safe operations for all other DRMAA methods.

```
void init(in string contactString)
   raises ( DrmsInitException,
            InvalidContactStringException,
            AlreadyActiveSessionException,
            DefaultContactStringException,
            NoDefaultContactStringSelectedException,
            OutOfMemoryException,
            DrmCommunicationException,
            AuthorizationException,
            InvalidArgumentException,
            InternalException);
```

Parameters

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

Exceptions

- DrmsInitException – failed while initializing the session.
- InvalidContactStringException – the *contact* parameter is invalid.
- AlreadyActiveSessionException – the session has already been initialized.
- DefaultContactStringException – the *contact* parameter is `null` or empty and the default contact string could not be used to connect to the DRMS.
- NoDefaultContactStringSelectedException – the *contact* parameter is `null` or empty and more than one DRMS is available.
- OutOfMemoryException – the DRMAA implementation does not have enough free memory to perform the operation.
- DrmCommunicationException – the DRMS could not be contacted for this request.
- AuthorizationException – the user does not have permission to perform this action.
- InvalidArgumentException – an argument value is invalid.
- InternalException – an error has occurred in the DRMAA implementation.

9.3 exit

The *exit()* method MUST do whatever work is required to disengage from the DRM system and allow the DRMAA implementation to perform any necessary internal cleanup. This method ends the current DRMAA session SHALL NOT affect any jobs (e.g., queued and running jobs remain queued and running). Any job template instances which have not yet been deleted become invalid after *exit()* is called, even after a subsequent call to *init()*. *exit()* SHOULD be called only once, by only one of the threads. Additional calls to *exit()* beyond the first SHALL throw a *NoActiveSessionException* or return a corresponding error code if exceptions aren't supported.

```
void exit()
   raises ( DrmsExitException,
            NoActiveSessionException,
            DrmCommunicationException,
            AuthorizationException,
            OutOfMemoryException,
            InternalException);
```

Exceptions

- DrmsExitException – failed while exiting the session.
- NoActiveSessionException – the session has not been initialized or *exit()* has already been called
- DrmCommunicationException – the DRMS could not be contacted for this request.
- AuthorizationException – the user does not have permission to perform this action.
- OutOfMemoryException – the DRMAA implementation does not have enough free memory to perform the operation.
- InternalException – an error has occurred in the DRMAA implementation.

9.4 createJobTemplate

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

```
JobTemplate createJobTemplate()
   raises ( DrmCommunicationException,
            NoActiveSessionException,
            OutOfMemoryException,
            AuthorizationException,
            InternalException);
```

Returns

The *createJobTemplate()* method SHALL return a blank *JobTemplate* instance.

Exceptions

- DrmCommunicationException – unable to communicate with the DRMS
- NoActiveSessionException – the session has not been initialized or *exit()* has already been called

- OutOfMemoryException – the DRMAA implementation does not have enough free memory to perform the operation.
- AuthorizationException – the user does not have permission to perform this action.
- InternalException – an error has occurred in the DRMAA implementation.

9.5 deleteJobTemplate

The *deleteJobTemplate()* method is used to deallocate a job template, and SHALL perform all necessary steps required to free all memory associated with the given JobTemplate instance.
In languages where memory is not freed explicitly, e.g. languages that use garbage collectors, this method SHALL perform all necessary steps required to prepare this job template to be freed. In languages where finalizers are supported, 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.*

```
void deleteJobTemplate(in DRMAA::JobTemplate jobTemplate)
  raises ( DrmCommunicationException,
           NoActiveSessionException,
           OutOfMemoryException,
           AuthorizationException,
           InvalidArgumentException,
           InvalidJobTemplateException,
           InternalException);
```

Parameters

`jobTemplate` - the *JobTemplate* instance to delete.

Exceptions

- DrmCommunicationException – unable to communicate with the DRMS.
- NoActiveSessionException – the session has not been initialized or *exit()* has already been called.
- OutOfMemoryException – the DRMAA implementation does not have enough free memory to perform the operation.
- AuthorizationException – the user does not have permission to perform this action.
- InvalidArgumentException – the argument value is invalid.
- InvalidJobTemplateException – the given job template was not created with *createJobTemplate()* or has already been deleted .
- InternalException – an error has occurred in the DRMAA implementation.

9.6 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.*

```
string runJob(in DRMAA::JobTemplate jobTemplate)
```

```
raises ( TryLaterException,
         DeniedByDrmException,
         DrmCommunicationException,
         AuthorizationException,
         InvalidJobTemplateException,
         NoActiveSessionException,
         OutOfMemoryException,
         InvalidArgumentException,
         InternalException);
```

Parameters

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

Returns

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

Exceptions

- 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.
- DrmCommunicationException – unable to communicate with the DRMS.
- InvalidJobTemplateException – the given job template was not created with *createJobTemplate()* or has already been deleted.
- AuthorizationException – the user does not have permission to submit jobs.
- NoActiveSessionException – the session has not been initialized or *exit()* has already been called.
- OutOfMemoryException – the DRMAA implementation does not have enough free memory to perform the operation.
- InvalidArgumentException – the argument value is invalid.
- InternalException – an error has occurred in the DRMAA implementation.

9.7 runBulkJobs

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

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

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

```
StringList runBulkJobs( in DRMAA::JobTemplate jobTemplate,
       in long beginIndex,
       in long endIndex,
       in long step)
  raises ( TryLaterException,
           DeniedByDrmException,
           DrmCommunicationException,
           AuthorizationException,
           InvalidJobTemplateException,
           NoActiveSessionException,
           OutOfMemoryException,
           InvalidArgumentException,
           InternalException);
```

Parameters

`jobTemplate` - the job template to be used to create the job.
`beginIndex` - the starting value for the loop index.
`endIndex` - the terminating value for the loop index.
`step` - the value by which to increment the loop index each iteration.

Returns

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

Exceptions

- 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.
- DrmCommunicationException – unable to communicate with the DRMS.
- InvalidJobTemplateException – the given job template was not created with *createJobTemplate()* or has already been deleted.
- AuthorizationException – the user does not have permission to submit jobs.
- NoActiveSessionException – the session has not been initialized or *exit()* has already been called.
- OutOfMemoryException – the DRMAA implementation does not have enough free memory to perform the operation.
- InvalidArgumentException – an argument value is invalid.
- InternalException – an error has occurred in the DRMAA implementation.

9.8 control

The *control()* method SHALL hold, release, suspend, resume, or kill the job identified by *jobName* respective to the *operation* parameter. The jobName parameter can be JOB_IDS_SESSION_ALL (see 9.1) to act on all jobs in the session.

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

The legal values for *operation* and their meanings SHALL be:

- `JobControlAction::SUSPEND`: stop the job,
- `JobControlAction::RESUME`: (re)start the job,
- `JobControlAction::HOLD`: put the job on-hold,
- `JobControlAction::RELEASE`: release the hold on the job, and
- `JobControlAction::TERMINATE`: kill the job.

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

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

```
void control( in string jobName,
     in JobControlAction operation)
   raises ( DrmCommunicationException,
            AuthorizationException,
            ResumeInconsistentStateException,
            SuspendInconsistentStateException,
            HoldInconsistentStateException,
            ReleaseInconsistentStateException,
            InvalidJobException,
            NoActiveSessionException,
            OutOfMemoryException,
            InvalidArgumentException,
            InternalException);
```

Parameters

`jobName` - The string id of the job to control.
`operation` - the control action to be taken.

Exceptions

- DrmCommunicationException – unable to communicate with the DRMS.
- AuthorizationException – the user does not have permission to modify jobs.
- ResumeInconsistentStateException – the job is not in a state from which is can be resumed.
- SuspendInconsistentStateException – the job is not in a state from which is can be suspended.
- HoldInconsistentStateException – the job is not in a state from which is can be held.
- ReleaseInconsistentStateException – the job is not in a state from which is can be released.
- InvalidJobException – the job id does not represent a valid job.
- NoActiveSessionException – the session has not been initialized or *exit()* has already been called.
- OutOfMemoryException – the DRMAA implementation does not have enough free memory to perform the operation.
- InvalidArgumentException – an argument value is invalid.

- InternalException – an error has occurred in the DRMAA implementation.

9.9 synchronize

This method SHALL wait until all jobs specified by *jobList* have finished execution. The jobList parameter can be JOB_IDS_SESSION_ALL (see 9.1) to act on all jobs in the session.

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

To prevent blocking indefinitely in this call, the caller may use a timeout specifying how many seconds to block in this call. The constant value *TIMEOUT_WAIT_FOREVER* may be specified to wait indefinitely for a result. The constant value *TIMEOUT_NO_WAIT* may be specified to return immediately. If the call exits before the timeout has elapsed, all the jobs have been waited on or there was an interrupt. If the invocation exits on timeout, an *ExitTimeoutException* SHALL be thrown or a corresponding error code returned if exceptions aren't supported. The caller should check system time before and after this call in order to be sure of how much time has passed.

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

The *dispose* parameter specifies how to treat the reaping of the remote job's internal data record, which includes a record of the job's consumption of system resources during its execution and other statistical information. If set to `true`, the DRM SHALL dispose of the job's data record. If set to `false`, the data record SHALL be left for future access via the *wait()* method. Because a DRMAA implementation is not required to retain information about jobs which have been reaped, the routine is not required to, but MAY distinguish between non-existent and reaped jobs. If the routine successfully validates a job ID for an already reaped job, it MAY return successfully without any error.

```
void synchronize(  in StringList jobList,
     in long long timeout,
     in boolean dispose)
   raises (  DrmCommunicationException,
             AuthorizationException,
             ExitTimeoutException,
             InvalidJobException,
             NoActiveSessionException,
             OutOfMemoryException,
             InvalidArgumentException,
             InternalException);
```

Parameters

`jobList` - the list of names for the jobs to synchronize.
`timeout` - the maximum number of seconds to wait.
`dispose` - specifies how to treat reaping information.

Exceptions

- DrmCommunicationException – unable to communicate with the DRMS.
- AuthorizationException – the user does not have permission to synchronize against jobs.
- 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.
- OutOfMemoryException – the DRMAA implementation does not have enough free memory to perform the operation.
- InvalidArgumentException – an argument value is invalid.
- InternalException – an error has occurred in the DRMAA implementation.

9.10 wait

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

The *timeout* value SHALL be used to specify the desired behavior when a result is not immediately available. The constant value `TIMEOUT_WAIT_FOREVER` may be specified to wait indefinitely for a result. The constant value `TIMEOUT_NO_WAIT` may be specified to return immediately. Alternatively, a number of seconds may be specified to indicate how long to wait for a result to become available.

If the call exits before *timeout* seconds, either the job has been waited on successfully or there was an abortion or termination of the job. If the invocation exits on timeout, an *ExitTimeoutException* SHALL be thrown or a corresponding error code returned if exceptions aren't supported. The caller should check system time before and after this call in order to be sure how much time has passed.

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

In a multi-threaded environment with a *wait()* call using `JOB_IDS_SESSION_ANY`, only the active thread gets the status of the finished or failed job in that case, while the other threads continue waiting. If there are no more running or completed jobs left in the session, all remaining waiting threads SHOULD fail with an *InvalidJobException*.

If thread A is waiting for a specific job, and another thread, thread B, waiting for that same job or with *JOB_IDS_SESSION_ANY,* receives notification that the job has finished, thread A SHOULD fail with an *InvalidJobException*. At any time during a call to *wait()* with *JOB_IDS_SESSION_ANY* as the *jobName* parameter, if no jobs are active in the session, the call to *wait()* SHALL fail, throwing an *InvalidJobException*.

When successful, the resource usage information for the job SHALL be provided as a `Dictionary` of usage parameter names and their values in the returned job info. The values contain the amount of resources consumed by the job and are implementation defined. If the resource usage information is unavailable, the provided Dictionary SHOULD be empty or null.

If the destination language does not support the notion of interfaces or objects, the wait() call SHOULD return an opaque data structure which contains the job exit information or references

to the job exit information. The opaque data structure is decoded using the routines which model the JobInfo interface.

```
JobInfo wait( in string jobName,
     in long long timeout)
   raises ( DrmCommunicationException,
            AuthorizationException,
            ExitTimeoutException,
            InvalidJobException,
            NoActiveSessionException,
            OutOfMemoryException,
            InvalidArgumentException,
            InternalException);
```

Parameters

`jobName` - the id of the job for which to wait.
`timeout` - the maximum number of seconds to wait.

Returns

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

Exceptions

- DrmCommunicationException – unable to communicate with the DRMS.
- AuthorizationException – the user does not have permission to wait for a job.
- 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.
- OutOfMemoryException – the DRMAA implementation does not have enough free memory to perform the operation.
- InvalidArgumentException – an argument value is invalid.
- InternalException – an error has occurred in the DRMAA implementation.

9.11 jobProgramStatus

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

- `JobProgramState:UNDETERMINED`: job status cannot be determined,
- `JobProgramState:QUEUED_ACTIVE`: job is queued and waiting to be scheduled,
- `JobProgramState:SYSTEM_ON_HOLD`: job has been placed on hold by the system or the administrator,
- `JobProgramState:USER_ON_HOLD`: job has been placed on hold by a user,
- `JobProgramState:USER_SYSTEM_ON_HOLD`: job has been placed on hold by both the system or administrator and a user,
- `JobProgramState:RUNNING`: job has been scheduled and is running,

- `JobProgramState:SYSTEM_SUSPENDED`: job has been suspended by the system or administrator,
- `JobProgramState:USER_SUSPENDED`: job has been suspended by a user,
- `JobProgramState:USER_SYSTEM_SUSPENDED`: job has been suspended by both the system or administrator and a user,
- `JobProgramState:DONE`: job finished normally, and
- `JobProgramState:FAILED`: job exited abnormally before finishing.

The DRMAA implementation MUST always get the status of the job from the DRM system unless the status has already been determined to be *`FAILED`* or *`DONE`* and the status has been successfully cached. Terminated jobs SHALL 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.

```
JobProgramState jobProgramStatus(in string jobName)
  raises ( DrmCommunicationException,
           AuthorizationException,
           InvalidJobException,
           NoActiveSessionException,
           OutOfMemoryException,
           InvalidArgumentException,
           InternalException);
```

Parameters

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

Returns

The *jobProgramStatus()* method SHALL return the program status.

Exceptions

- DrmCommunicationException – unable to communicate with the DRMS.
- AuthorizationException – the user does not have permission to query for a job's status.
- InvalidJobException – the job id does not represent a valid job.
- NoActiveSessionException – the session has not been initialized or *exit()* has already been called.
- OutOfMemoryException – the DRMAA implementation does not have enough free memory to perform the operation.
- InvalidArgumentException – an argument value is invalid.
- InternalException – an error has occurred in the DRMAA implementation.

9.12 contact

If this attribute is read before the first call to the *init()* method, then it SHALL return a string containing a comma-delimited list of default DRMAA implementation contacts strings. A contact string represents a specific installation of a specific DRM system, e.g. a Condor central manager machine at a given IP address or a Sun Grid Engine 'root' and 'cell'.
If the value of the attribute is queried after a successful call to *init()*, this attribute SHALL contain 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.

```
readonly attribute string contact;
```

9.13 version

This attribute SHALL contain a *Version* instance containing the major and minor version numbers of the DRMAA library. This attribute may not be read before *init()* has been called.

```
readonly attribute DRMAA::Version version;
```

9.14 drmsInfo

If the value of this attribute is read before the first successful call to the *init()* method, this attribute SHALL return a string containing a comma-delimited list of DRM system identifiers. A DRM system identifier denotes a specific type of DRM system, e.g. Sun Grid Engine.
If the value is read after *init()*, this attribute SHALL contain the selected DRM system identifier. The Strings are implementation dependent and SHOULD NOT be interpreted by the application.

```
readonly attribute string drmsInfo;
```

9.15 drmaaImplementation

If the value of this attribute is read before the first successful call to *init()*, this attribute SHALL return a string containing a comma-delimited list of DRMAA implementations. A DRMAA implementation string denotes a specific version of a DRM system, e.g. Condor v6.6.
If read after *init()*, this attribute SHALL contain the selected DRMAA implementation.
The returned Strings are implementation dependent and SHOULD NOT be interpreted by the application.

```
readonly attribute string drmaaImplementation;
```

10 Annex

10.1 Complete IDL specification

```
module DRMAA{
   // unbounded native ordered string list
   valuetype OrderedStringList sequence<string>;
   // unbounded native string list
   valuetype StringList sequence<string>;
   // dictionary type, for unbounded key-value pair storage
   valuetype Dictionary sequence< sequence<string,2> >;
   // amount of time, at least with a resolution to seconds
   valuetype TimeAmount long long;
   enum JobControlAction {
      SUSPEND,
      RESUME,
      HOLD,
      RELEASE,
      TERMINATE
   };
   enum JobProgramState {
      UNDETERMINED,
      QUEUED_ACTIVE,
      SYSTEM_ON_HOLD,
      USER_ON_HOLD,
      USER_SYSTEM_ON_HOLD,
      RUNNING,
      SYSTEM_SUSPENDED,
      USER_SUSPENDED,
      USER_SYSTEM_SUSPENDED,
      DONE,
      FAILED
   };

   enum JobSubmissionState {
             HOLD_STATE,
             ACTIVE_STATE
   };
   valuetype FileTransferMode {
      attribute boolean transferInputStream;
      attribute boolean transferOutputStream;
      attribute boolean transferErrorStream;
   };
   valuetype Version {
      readonly attribute long major;
      readonly attribute long minor;
   };
   exception AlreadyActiveSessionException {string message;};
   exception AuthorizationException {string message;};
   exception ConflictingAttributeValuesException {string message;};
   exception DefaultContactStringException {string message;};
   exception DeniedByDrmException {string message;};
   exception DrmCommunicationException {string message;};
   exception DrmsExitException {string message;};
   exception DrmsInitException {string message;};
   exception ExitTimeoutException {string message;};
   exception HoldInconsistentStateException {string message;};
```

```
exception IllegalStateException {string message;};
exception InternalException {string message;};
exception InvalidArgumentException {string message;};
exception InvalidAttributeFormatException {string message;};
exception InvalidAttributeValueException {string message;};
exception InvalidContactStringException {string message;};
exception InvalidJobException {string message;};
exception InvalidJobTemplateException {string message;};
exception NoActiveSessionException {string message;};
exception NoDefaultContactStringSelectedException {string message;};
exception OutOfMemoryException {string message;};
exception ReleaseInconsistentStateException {string message;};
exception ResumeInconsistentStateException {string message;};
exception SuspendInconsistentStateException {string message;};
exception TryLaterException {string message;};
exception UnsupportedAttributeException {string message;};
native PartialTimestamp;
interface JobInfo {
   readonly attribute string jobId;
   readonly attribute Dictionary resourceUsage;
   readonly attribute boolean hasExited;
   readonly attribute long exitStatus;
   readonly attribute boolean hasSignaled;
   readonly attribute string terminatingSignal;
   readonly attribute boolean hasCoreDump;
   readonly attribute boolean wasAborted;

};
interface JobTemplate{
   const string HOME_DIRECTORY = "$drmaa_hd_ph$";
   const string WORKING_DIRECTORY = "$drmaa_wd_ph$";
   const string PARAMETRIC_INDEX = "$drmaa_incr_ph$";
   attribute string remoteCommand;
   attribute OrderedStringList args;
   attribute DRMAA::JobSubmissionState jobSubmissionState;
   attribute Dictionary jobEnvironment;
   attribute string workingDirectory;
   attribute string jobCategory;
   attribute string nativeSpecification;
   attribute StringList email;
   attribute boolean blockEmail;
   attribute PartialTimestamp startTime;
   attribute string jobName;
   attribute string inputPath;
   attribute string outputPath;
   attribute string errorPath;
   attribute boolean joinFiles;
   attribute FileTransferMode transferFiles;
   attribute PartialTimestamp deadlineTime;
   attribute TimeAmount hardWallclockTimeLimit;
   attribute TimeAmount softWallClockTimeLimit;
   attribute TimeAmount hardRunDurationLimit;
   attribute TimeAmount softRunDurationLimit;
   readonly attribute StringList attributeNames;
      …
   [language-specific operations for implementation-specific attributes]
      …
interface Session{
  const long long TIMEOUT_WAIT_FOREVER = -1;
  const long long TIMEOUT_NO_WAIT = 0;
```

```
const string JOB_IDS_SESSION_ANY = "DRMAA_JOB_IDS_SESSION_ANY";
const string JOB_IDS_SESSION_ALL = "DRMAA_JOB_IDS_SESSION_ALL";
void init(in string contactString)
   raises ( DrmsInitException,
            InvalidContactStringException,
            AlreadyActiveSessionException,
            DefaultContactStringException,
            NoDefaultContactStringSelectedException,
            OutOfMemoryException,
            DrmCommunicationException,
            AuthorizationException,
            InvalidArgumentException,
            InternalException);
void exit()
   raises ( DrmsExitException,
            NoActiveSessionException,
            DrmCommunicationException,
            AuthorizationException,
            OutOfMemoryException,
            InternalException);
JobTemplate createJobTemplate()
   raises ( DrmCommunicationException,
            NoActiveSessionException,
            OutOfMemoryException,
            AuthorizationException,
            InternalException);
void deleteJobTemplate(in DRMAA::JobTemplate jobTemplate)
   raises ( DrmCommunicationException,
            NoActiveSessionException,
            OutOfMemoryException,
            AuthorizationException,
            InvalidArgumentException,
            InvalidJobTemplateException,
            InternalException);
string runJob(in DRMAA::JobTemplate jobTemplate)
   raises ( TryLaterException,
            DeniedByDrmException,
            DrmCommunicationException,
            AuthorizationException,
            InvalidJobTemplateException,
            NoActiveSessionException,
            OutOfMemoryException,
            InvalidArgumentException,
            InternalException);
StringList runBulkJobs( in DRMAA::JobTemplate jobTemplate,
       in long beginIndex,
       in long endIndex,
       in long step)
   raises ( TryLaterException,
            DeniedByDrmException,
            DrmCommunicationException,
            AuthorizationException,
            InvalidJobTemplateException,
            NoActiveSessionException,
            OutOfMemoryException,
            InvalidArgumentException,
            InternalException);
void control( in string jobName,
     in JobControlAction operation)
```

```
    raises ( DrmCommunicationException,
             AuthorizationException,
             ResumeInconsistentStateException,
             SuspendInconsistentStateException,
             HoldInconsistentStateException,
             ReleaseInconsistentStateException,
             InvalidJobException,
             NoActiveSessionException,
             OutOfMemoryException,
             InvalidArgumentException,
             InternalException);

  void synchronize(  in StringList jobList,
      in long long timeout,
      in boolean dispose)
    raises ( DrmCommunicationException,
             AuthorizationException,
             ExitTimeoutException,
             InvalidJobException,
             NoActiveSessionException,
             OutOfMemoryException,
             InvalidArgumentException,
             InternalException);
  JobInfo wait( in string jobName,
      in long long timeout)
    raises ( DrmCommunicationException,
             AuthorizationException,
             ExitTimeoutException,
             InvalidJobException,
             NoActiveSessionException,
             OutOfMemoryException,
             InvalidArgumentException,
              InternalException);

  JobProgramState jobProgramStatus(in string jobName)
    raises ( DrmCommunicationException,
             AuthorizationException,
             InvalidJobException,
             NoActiveSessionException,
             OutOfMemoryException,
             InvalidArgumentException,
             InternalException);
  readonly attribute string contact;
  readonly attribute DRMAA::Version version;
  readonly attribute string drmsInfo;
  readonly attribute string drmaaImplementation;
```

```
                };
    };
```

10.2 Correlation of DRMAA exceptions and error codes

The following table shows how exceptions can map to error codes, similar to the definition in the Distributed Resource Management Application API Specification 1.0 [DRMAA10].

Error Code Name (DRMAA_ERRNO_...)	***Exception Name***
SUCCESS	*Not needed*
INTERNAL_ERROR	InternalException
DRM_COMMUNICATION_FAILURE	DrmCommunicationException
AUTH_FAILURE	AuthorizationException
INVALID_ARGUMENT	InvalidArgumentException
NO_ACTIVE_SESSION	NoActiveSessionException
NO_MEMORY	OutOfMemoryException
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	ConflictingAttributeValuesException
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

Error Code Name (DRMAA_ERRNO_...)	*Exception Name*
NO_RUSAGE	*Not needed*
INVALID_JOB_TEMPLATE	InvalidJobTemplateException
UNSUPPORTED_ATTRIBUTE	UnsupportedAttributeException

The DRMAA_ERRNO_SUCCESS code reflects a successful operation call, if a language binding models the error codes as operation return values. The DRMAA_ERRNO_NO_RUSAGE is used to indicate that the target of a wait() call has exited without providing resource usage information in languages which do no support the notion of interfaces or objects. See section 9.

In comparison to [DRMAA10], this specification introduces two new error conditions. The *InvalidJobTemplateException* is used to indicate that the job template instance currently being used is not valid. This may be, for example, because it has already been deleted via *Session::deleteJobTemplate()*. The *UnsupportedAttributeException* is used to indicate that for the current DRMAA implementation the accessed attribute of a job template is unsupported.

10.3 Correlation of *JobTemplate* attributes and attribute name strings

The following table shows the string names for the attributes in the *JobTemplate* interface. The string names are needed as input parameter for the *JobTemplate.getAttribute()* and *JobTemplate.setAttribute()* operations (see Section 8.1.1. Following the [DRMAA10] semantics, *JobTemplate* attributes with a complex type are prefixed by “v_” (vector attribute).

String Name	*JobTemplate Attribute*
“remote_command”	JobTemplate.remoteCommand
“v_argv”	JobTemplate.args
“js_state”	JobTemplate.jobSubmissionState
“v_env”	JobTemplate.jobEnvironment
“wd”	JobTemplate.workingDirectory
“job_category”	JobTemplate.jobCategory
“native_specification”	JobTemplate.nativeSpecification
“v_email”	JobTemplate.email
“block_email”	JobTemplate.blockEmail
“start_time”	JobTemplate.startTime
“job_name”	JobTemplate.jobName
“input_path”	JobTemplate.inputPath

String Name	*JobTemplate Attribute*
“output_path”	JobTemplate.outputPath
“error_path”	JobTemplate.errorPath
“join_files”	JobTemplate.joinFiles
“transfer_files”	JobTemplate.transferFiles
“deadline_time”	JobTemplate.deadlineTime
“wct_hlimit”	JobTemplate.hardWallclockTimeLimit
“wct_slimit”	JobTemplate.softWallclockTimeLimit
“run_duration_hlimit”	JobTemplate.hardRunDurationLimit
“run_duration_slimit”	JobTemplate.softRunDurationLimit

11 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.

12 References

[OMG IDL] Object Management Group. Common Object Request Broker Architecture: Core Specification, Chapter 3, March 2004

[RFC 2119] S. Bradner. RFC 2119 – Key words for use in RFCs to Indicate Requirement Levels, March 1997

[DRMAA10] Hrabri Rajic et.al. Distributed Resource Management Application API Specification 1.0 (GFD.022). June 2004

13 Contributors

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
18 Network Circle, UMPK18-117
Menlo Park, CA 94025

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

14 Acknowledgements

We are grateful to numerous colleagues for support and discussions on the topics covered in this document, in particular (in alphabetical order, with apologies to anybody we've missed) Guillaume Alleon, Ali Anjomshoaa, Ed Baskerville, Harald Böhme, Matthieu Cargnelli, Karl Czajkowski, Paul Foley, Nicholas Geib, Becky Gietzel, Ancor Gonzalez Sosa, Tim Harsch, Greg Hewgill, Rayson Ho, Eduardo Huedo, Dieter Kranzmüller, Peter G. Lane, Miron Livny, Ignacio M. Llorente, Martin v. Löwis, Andre Merzky, Ruben S. Montero, Greg Newby, Steven Newhouse, Michael Primeaux, Greg Quinn, Martin Sarachu, Jennifer Schopf, Enrico Sirola, Chris Smith, Douglas Thain, Jose R. Valverde, and Peter Zhu.

15 Intellectual Property Statement

16 Disclaimer

17 Full Copyright Notice