Distributed Resource Management
Application API (DRMAA)
Working Group

Peter Tröger, Humboldt University (editor)
Daniel Templeton, Sun Microsystems
Hrabri Rajic, Intel Americas Inc.
Roger Brobst, Cadence Design Systems

Distributed Resource Management Application API 2.0

Peter Tröger 3.9.08 13:59
Kommentar: "TODO" marks open issues, "RAT" a rationale for changes. TODO items arose from experience reports, the survey results, and tracker items.

Peter Tröger 3.9.08 11:18
Kommentar: According to the survey, most people can live with non-backward compatible changes

Peter Tröger 3.9.08 13:59
Kommentar: TODO: According to survey, DRMAA2 should be aligned to OGSA-BES, SAGA, and Windows HPC

Peter Tröger 3.9.08 13:59
Kommentar: TODO: #6275 – Define all default values.

Status of This Document

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

Copyright Notice

Abstract

This document describes the common base for the Distributed Resource Management Application API v2.0 (DRMAA) bindings for procedural and object-oriented languages.

Table of Contents

… (LEFT OUT FOR EASIER CHANGE TRACKING) …

1 Introduction

This document gives an IDL description for the DRMAA interface. The specification provided by this document is completely language-independent, even though some of the examples are given in Java. Adopters of this specification are expected to derive a language-binding specification (as described in Section 2.2), which can then be centrally published by the DRMAA working group. This ensures portability for DRMAA applications in one programming language, and ensures consistent API semantics over all possible DRMAA language bindings.

Peter Tröger 3.9.08 14:04
Kommentar: TODO: Describe relation to GFD.130 / 133

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.

1.2   Related Work

There are other relevant OGF standards in the area of job submission and monitoring. An in-depth comparison and positioning of DRMAA v1.0 is provided by a conference publication [IJGUC08].

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 DRMAA 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.

Job identifiers in the DRMAA specification are represented as Strings, instead of introducing an own *Job* object. Even though this could enable a tighter relation of job meta-data and identity, it mainly increases complexity for persistent sessions and introduces unnecessary round trips in a possible DRMAA RPC mapping (such as a WSDL binding). The representation of jobs as string directly reflects the DRM system semantic, and therefore supports the intentional tight binding of DRMAA application and execution environment.

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.

> Peter Tröger 3.9.08 13:58
> **Kommentar:** TODO: #6277 – Relax this formulation to ease up the Python binding.

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^TM 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

The DRMAA specification 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 Changes in comparison to GFD.130 (DRMAA 1.0)

- The differentiation between the system hold, user hold, and system / user hold job states was removed (conf. call Jan 20th 2009). There is only one hold state now.
- A job can now change its state from one of the SUSPENDED states to the QUEUED_ACTIVE state (conf. call Jan 20th 2009, solves issue #2788).
- The job state UNDETERMINED is now clearer defined. It expressed a permanent issue, meaning that the job state will not change by just waiting. Temporary problems in the detection of the job state are now expressed by the TryLaterException (conf. call Feb 5th 2009, solves issue #2783).
- The concept of a factory in GFD.130 was removed (solves issue #6276).
- The jobState() function now also returns job *subState* information. This is intended as additional information for the given DRMAA job state, and can be used for expressing the hold state differentiation from DRMAA 1.0 (conf. call Mar 31st 2009).
- The *PartialTimestamp* functionality was completely removed. Absolute date and time values are now expressed as RFC822 conformant string (conf. call Mar 31st 2009).
- Beside the *FAILED* state from DRMAA 1.0, there is now also the *TERMINATED* state for jobs that did not finish execution successfully (conf. call XXXX – ist schon geändert, muss aber noch bestätigt werden). Due to the more specific differentiation, all runtime job template errors (such as an incorrect working directory) now lead to *TERMINATED* instead of *FAILED* state. *JobInfo:terminatingSignal* now only provides useful information in case of the *FAILED* state.
- The *JobInfo* structure was extended to contain also the job state at the time of querying. This is needed to interpret exit code and signal information correctly with the new differentiation between TERMINATED and FAILED state (conf. call Mar 31st 2009).

4 The DRMAA2 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 DRMAA2{
    ...
}
```

> Peter Tröger 3.9.08 09:36
> **Kommentar:** RAT: This intentionally breaks backward compatibility for existing apps.

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 DRMAA2	package org.drmaa2

5 Data Types

5.1 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
};
```

5.2 JobState enumeration

The *JobState* enumeration is used as return value type for the *jobStatus()* 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 language binding implementation is not required to be able to return all of the job state values in the *JobState* 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 *JobState* 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 *JobState* enumeration.

```
enum JobState {
   UNDETERMINED,
   QUEUED_ACTIVE,
   HOLD,
   RUNNING,
   SYSTEM_SUSPENDED,
   USER_SUSPENDED,
   USER_SYSTEM_SUSPENDED,
   DONE,
   FAILED,
   TERMINATED
};
```

The status values relate to the DRMAA job state transition model:

```
Valuator -> Queued_Active
Valuator -> Hold
Valuator -> Rejected
Queued_Active <-> Hold
Queued_Active -> Running
Queued_Active -> Terminated
Hold -> Terminated
Running <-> System_Suspended
Running <-> User_Suspended
Running <-> User-System_Suspended
Running -> Failed
Running -> Terminated
System_Suspended -> Failed
System_Suspended -> Terminated
User_Suspended -> Failed
User_Suspended -> Terminated
User-System_Suspended -> Failed
User-System_Suspended -> Terminated
System_Suspended -> Queued_Active
User_Suspended -> Queued_Active
User-System_Suspended -> Queued_Active
Running -> Done
Undetermined
```

Figure 1: DRMAA Job State Transition Diagram

5.3 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
};
```

5.4 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;
};
```

5.4.1 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.

5.4.2 transferOutputStream

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

5.4.3 transferErrorStream

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

5.5 Version value type

The *Version* value type is a holding structure for the major and minor version numbers of the DRMAA language binding 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;
};
```

5.5.1 major

This attribute SHALL contain the major version number.

5.5.2 minor

This attribute SHALL contain the minor version number.

6 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 (like Java), 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. Such a language binding SHOULD specify numeric values for all DRMAA error constants.

6.1 AlreadyActiveSessionException

Initialization failed due to existing DRMAA session.

6.2 AuthorizationException

The user is not authorized to perform the given operation.

6.3 ConflictingAttributeValuesException

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

6.4 DefaultContactStringException

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

6.5   DeniedByDrmException

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

6.6   DrmCommunicationException

Could not contact DRM system.

6.7   DrmsExitException

A problem was encountered while trying to exit the session.

6.8   DrmsInitException

A problem was encountered while trying to initialize the session.

6.9   ExitTimeoutException

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

6.10   HoldInconsistentStateException

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

6.11   InternalException

An unexpected or internal DRMAA error occurred, for example a system call failure.

6.12   InvalidArgumentException

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

6.13   InvalidAttributeFormatException

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

6.14   InvalidAttributeValueException

The value for the job template property is invalid.

6.15   InvalidContactStringException

The given contact string is not valid.

6.16   InvalidJobException

The job specified by the given job id does not exist, or was already reaped by a call to *Session::synchronize()* with *dispose==TRUE*.

6.17 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.

6.18 NoActiveSessionException

The method call failed because there is no active session.

6.19 NoDefaultContactStringSelectedException

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

6.20 OutOfMemoryException

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

6.21 ReleaseInconsistentStateException

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

6.22 ResumeInconsistentStateException

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

6.23 SuspendInconsistentStateException

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

6.24 TryLaterException

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

6.25 UnsupportedAttributeException

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

6.26 IllegalStateException

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

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:

> Peter Tröger 3.9.08 13:27
> **Kommentar:** TODO: #5874 – replace POSIX-style status check by something with one-call semantics; proposals in tracker history and GFD.117

```
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;
    readonly attribute JobState jobState;

};
```

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

This attribute provides the identifier of 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.

The user MAY reap this data only once. The implementation is free to "garbage collect" the reaped data at a convenient time. Only the data from the current session's job Id MUST be available. Reaping data from other session job Id's MAY be supported in a DRMAA implementation.

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 in FAILED state 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, or that the job is not in FAILED state. In all these cases *terminatingSignal* SHALL NOT provide signal information.

7.6 terminatingSignal

If *hasSignaled* is *true and the job state is FAILED*, this attribute SHALL contain a representation of the signal that caused the termination of the job's operation system process. 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.

7.9 jobState

This attribute SHALL contain the actual job state from the time were the *JobInfo* information was collected.

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 then 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 [GFD133] section 3.1.4 to 3.1.6 for more information regarding precedence rules, site-specific requirements and job evaluation.

8.1 Interface overview

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 string startTime;
    attribute string jobName;
    attribute string inputPath;
```

```
    attribute string outputPath;
    attribute string errorPath;
    attribute boolean joinFiles;
    attribute FileTransferMode transferFiles;
    attribute string 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 that 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 strings 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 that 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 that 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 that 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 of writing fully dynamic, introspection-based application logic.

8.2.2 Dynamic Loader Approach
In languages that 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 that 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.

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 that 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 placeholder 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 placeholder used to represent the current working directory when building paths for the *inputPath*, *outputPath*, and *errorPath* attributes.

The *PARAMETRIC_INDEX* constant is a placeholder 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

This attribute describes the command to 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

This attribute contains the list of command-line arguments for the job to be executed.

> **Peter Tröger 3.9.08 10:40**
> **Kommentar:** TODO: #2837 – more placeholders. Was favored by most survey participants. Needs research about common placeholders in today's DRM systems.

> **Peter Tröger 3.9.08 14:09**
> **Kommentar:** TODO: #5873 – support for the placeholders in more of the JT attributes. Needs research about DRM support. Some parts might be implementable in the DRMAA library only.

8.6 jobSubmissionState

Defines the state of the job at submission time. For more information see section 5.3.

8.7 jobEnvironment

This attribute holds the environment variable values for the execution machine. The values SHOULD 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 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 *JobState.TERMINATED*.

8.9 jobCategory

This attribute allows 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

This attribute enables 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

This attribute holds a list of email addresses that is used to report the job completion and status.

Peter Tröger 1.4.09 10:38
Gelöscht: *FAILED*

Peter Tröger 3.9.08 14:10
Kommentar: TODO: #5853 – Standardize category names. Otherwise check if this attribute was ever implemented.

Peter Tröger 3.9.08 13:24
Kommentar: TODO: Make document self-contained.

Peter Tröger 3.9.08 13:25
Kommentar: TODO: make document self-contained.

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. Date and time are expressed as RFC822 conformant string.

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 were successfully submitted, and the file can't be read, the job enters the state, *JobState.TERMINATED*.

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 were successfully submitted and the file can't be written before execution the job MUST enter the state `JobState.`*`TERMINATED`*.

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 were successfully submitted and the file can't be written before execution, the job enters the state *JobState.TERMINATED*.

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. Date and time are expressed as RFC822 conformant string.

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 section 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.

An application process SHALL open only one DRMAA session at a time. Another session can be opened only after the current one is closed. Nesting of sessions SHOULD NOT be possible. Job Id's SHALL remain valid from one session to another.

An implementation MAY persist job ID's of submitted jobs for the next session, in order to support a restartable application (e.g. Java servlets). The behavior of those jobs in the subsequent session is undefined. Job control routines MAY work correctly if a job ID was generated in a previous DRMAA session provided the current DRMAA session knows how to resolve this job ID. The burden is on the user to match previous job Id's with appropriate DRMAA sessions (i.e., DRMAA implementations).

Without session persistency, it is RECOMMENDED that the DRMAA library free all the session resources, although this is not guaranteed, so that old session resources cannot be used later.

The Session interface has explicit methods for starting and ending the session, as well as for creating and destroying job template objects. Even though some object oriented programming languages might prefer implicit object destruction mechanism instead of explicit cleanup calls, this interface design reflects the close coupling of DRMAA to the underlying DRM system. It also supports the implementation of object oriented DRMAA libraries based on a DRMAA C library.

```
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, TERMINATED* 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, TERMINATED* 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);
```

Peter Tröger 3.9.08 13:38
Kommentar: TODO: #2822 – improve description of contact string parameter

Peter Tröger 3.9.08 14:13
Kommentar: TODO: Python people hate that – this is what parameterized constructors are good for.

`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.

> Peter Tröger 3.9.08 14:14
> **Kommentar:** TODO: Python people hate that – this is what finalizers are good for.

```
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);
```

<u>Parameters</u>

`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.

<u>Returns</u>

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

<u>Exceptions</u>

- 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 section 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 the parameter is 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 that 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 unfinished 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   jobStatus

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

- `JobState:UNDETERMINED`: The job status cannot be determined. This is a permanent issue, not being solvable by querying again for the job state.
- `JobState:QUEUED_ACTIVE`: The job is queued for being scheduled and executed.
- `JobState:HOLD`: The job has been placed on hold by the system, the administrator, or the user.
- `JobState:RUNNING`: The job is running in the DRM system.
- `JobState:SYSTEM_SUSPENDED`: The job has been suspended by the system or the administrator.
- `JobState:USER_SUSPENDED`: The job has been suspended by a user.
- `JobState:USER_SYSTEM_SUSPENDED`: The job has been suspended by both the system or administrator and a user.
- `JobState:DONE`: The job finished without an error.

- `JobState:FAILED`: The job exited abnormally by itself before finishing. In POSIX-like systems, this is typically triggered by a signal to the application process.

- `JobState:TERMINATED`: The job was ended by an external entity (e.g. the user or the DRM system) before finishing execution. In POSIX-like systems, this is typically triggered by a signal 'to the job' containing the application process.

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*, *TERMINATED* or *DONE* and the status has been successfully cached. It is up to the implementation to determine whether this method is capable of operating on jobs submitted outside of the current DRMAA session.

```
JobState jobStatus(in string jobName, out native subState)
raises ( DrmCommunicationException,
         AuthorizationException,
         InvalidJobException,
         NoActiveSessionException,
         OutOfMemoryException,
         InvalidArgumentException,
         InternalException);
```

Parameters

`jobName` - the id of the job whose status is to be retrieved.
subState – a DRMS-specific sub state for the returned state

Returns

The jobStatus() method SHALL return the job status, together with an implementation specific sub state. This is intended to be a more detailed description of the current DRMAA job state, for example the specific kind of HOLD state (user-triggered, system-triggered). Applications SHOULD NOT expect this information to be available in all cases. Language bindings MUST allow the application to discard this information (e.g. by passing a NULL value), and SHOULD use a generic reference data type (e.g. *void or Object pointer). Implementations of the DRMAA API SHOULD then define a DRMS-specific data structure for the sub-state information.

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

… (LEFT OUT FOR EASIER CHANGE TRACKING) …

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 [GFD133].

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

Error Code Name (DRMAA_ERRNO_...)	Exception Name
HOLD_INCONSISTENT_STATE	HoldInconsistentStateException
RELEASE_INCONSISTENT_STATE	ReleaseInconsistentStateException
EXIT_TIMEOUT	ExitTimeoutException
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 7.

In comparison to [GFD133], 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 [GFD133] 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

String Name	JobTemplate Attribute
"block_email"	JobTemplate.blockEmail
"start_time"	JobTemplate.startTime
"job_name"	JobTemplate.jobName
"input_path"	JobTemplate.inputPath
"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.

Peter Tröger 3.9.08 11:10

Kommentar: TODO: According to survey, some people want us to consider security details, such as the user account the job is running under. Needs check with both the DRM systems and GridWay.

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

[IJGUC08]   Peter Tröger, Hrabri Rajic, Andreas Haas, and Piotr Domagalski. Standardized Job Submission and Control in Cluster and Grid Environments. In International Journal of Grid and Utility Computing (IJGUC). 2008. ISSN 1741-847X

[GFD133]   Hrabri Rajic, Roger Brobst, Waiman Chan, Fritz Ferstl, Jeff Gardiner, Andreas Haas, Bill Nitzberg, John Tollefsrud, and Peter Tröger. Distributed Resource Management Application API Specification 1.0 (GFD.133). Grid Recommendation. Open Grid Forum, 2008.

13 Contributors

Peter Tröger

peter@troeger.eu
Blekinge Institute of Technology

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, Piotr Domagalski, Fritz Ferstl, Paul Foley, Nicholas Geib, Becky Gietzel, Alleon Guillaume, Tim Harsch, Greg Hewgill, Rayson Ho, Eduardo Huedo, Dieter Kranzmüller, Krzysztof Kurowski, 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, Ancor Gonzalez Sosa, Douglas Thain, John Tollefsrud, Jose R. Valverde, and Peter Zhu.

15 Intellectual Property Statement

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

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

16 Disclaimer

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

17 Full Copyright Notice

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