Home

Welcome to DRMAAv2

DRMAA is an Open Grid Forum (OGF) API specification for the submission and control of jobs to one or more Distributed Resource Management (DRM) systems. It is the primary programming API for Sun Grid Engine, but also available for other DRM systems.

The official DRMAA home is http://www.drmaa.org/. This Wiki serves as discussion and collaboration platform for the upcoming new version of the DRMAA specification.

Everybody is invited to post comments about the different parts of the new API. Your comments will be discussed on the DRMAA mailing list and during the phone conferences - see http://www.drmaa.org/ for further contact details.

Discussion Pages

Suggested Configuration Names

Table of Content

Collapse all
Expand all   Collapse all

Recently Updated

Data Types	by troeger (14 Dec)
statemodelv2.png	by troeger (14 Dec)
OpenIssues	by troeger (12 Dec)
JobSession	by troeger (12 Dec)
SessionManager	by templedf (12 Dec)
DRMAAv2 API	by troeger (12 Dec)
Exceptions	by templedf (12 Dec)
MonitoringSession	by templedf (12 Dec)
JobTemplate	by templedf (12 Dec)
JobInfo	by templedf (12 Dec)
ReservationInfo	by mmamonski (25 Aug)
Reservation	by mmamonski (25 Aug)
Changes in comparison to GFD.130	by troeger (25 Aug)
Re: DRMAAv2 API	by troeger (27 Jul)
DrmaaCallback	by troeger (13 Jul)
Re: DRMAAv2 API	by dg222156 (09 Jul)
Re: JobTemplate	by dg222156 (09 Jul)

Introduction	by troeger (07 Jul)
Re: JobTemplate	by troeger (06 Jul)
Home	by troeger (06 Jul)

Editor Manual

Editors of the Specification Text should follow these rules, in order to allow an easier Word document generation in the finalization phase:

- IDL or programming language keywords should be formatted as fixed-with text with 'two-curly-brackets' markup.
- Open issues that are not related to a particular piece of text should be stored on the DRMAAv2 API page.
- Open issues for particular text should be realized with the note markup: `{note:title=issuetext}bla{note}`
- Section titles should start from `\h1`. Usually, the root-level title (== chapter title) is the same as the Wiki page title.

OpenIssues

- Standardize keys in resourceUsage attribute of JobInfo (Dan).
- Resolve how to report slot assignments for jobs (Peter).
- Finish JobSession chapter text (Peter).
- Prettify the state diagram in DataTypes chapter (Peter).
- Finish SessionManager chapter text (Dan).
- Check interface, fix chapter text:
- DrmaaCallback, (Dan)
- Reservation, (Peter)
- ReservationTemplate, (Peter)
- ReservationInfo, (Peter)
- ReservationSession, (Peter)
- Job. (Dan)
- Global check for error conditions and according exceptions. (Peter)
- Move to OGF document. (Peter)
- Language fixing (Dan).

Specification Text

Distributed Resource Management Application API 2.0

Editors

- Peter Tröger, Hasso-Plattner-Institute
- Daniel Templeton, Sun Microsystems

Status of This Document

This document provides information to the Grid community. Distribution is unlimited. This document obsoletes GFD-133.

Copyright Notice

Abstract

Abstract

This document describes the Distributed Resource Management Application API (DRMAA) v2.0, which provides a generalized API to distributed resource management systems (DRMS) in order to facilitate integration of application programs.

The scope of DRMAA is limited to job submission, job control, and retrieval of job and machine monitoring information. DRMAA provides application developers and high-level API designers with a programming model that allows a tightly coupled, but portable access to an underlying DRMS.

This document describes the common base for procedural and object-oriented language bindings. The intended audience are DRMAA language binding designers, DRMS vendors, high-level API designers and meta-scheduler architects. End users are expected to rely on product-specific documentation for the DRMAA API.

Introduction

This document describes an API for job submission, job control, and monitoring in Distributed Resource Management Systems (DRMS). The objective is to facilitate the direct interfacing of applications to DRMS for application builders, portal builders, and independent software vendors. The specification abstracts the fundamental interfaces of DRMS and provides an easy-to-use programming model, thereby encouraging adoption by both application builders and DRMS builders.

The document describes the DRMAA interface semantics with the help of OMG IDL. Adopters of this specification are expected to derive a language-binding specification, which can then be centrally approved by the DRMAA working group. While this document has the responsibility to ensure consistent API semantics over all possible DRMAA implementations, the language binding has the responsibility to ensure source-code portability for DRMAA applications between different DRM systems. An effort has been made to choose an API layout that is 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. Some parts of the document include rules and suggestions for the creation of specific language bindings. Specific examples are given in the Java language. These examples are not normative.

This document describes the second version of the DRMAA API. The first version of DRMAA, as described in GFD.22 and GFD.133, is obsoleted by this specification. A later secion describes the API design changes in the new version, in order to ease the porting of existing implementations.

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

Notational Conventions

In this document, IDL resp. programming language elements and definitions are represented in a `fixed-width font`. 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.

The following abbreviations are used in this document:

API	Application Programming Interface
DRMS	Distributed Resource Management System, a cluster or Grid infrastructure being interfaced through DRMAA
DRM	Distributed Resource Management
DRMAA	Distributed Resource Management Application API

Designing a language binding

Language binding documents 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, since they have a huge overhead of irrelevant CORBA-related mapping rules. Therefore, one must carefully decide if a binding decision reflects a natural and simple mapping of the intended purpose for the DRMAA interfaces. The binding SHOULD reuse OMG value type mappings (e.g. IDL `long long` to Java `long`), and SHOULD define custom mappings for the other types. However, the language binding MUST use the described concept mapping in a consistent manner for its overal API layout.

Due to the usage of IDL, all method groups for a particular purpose (e.g. job control) are described in terms of interfaces, and not classes. The usage of a class concept depends on the specific language-mapping rules.

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

Access to scalar attributes (`string, Boolean, long`) MUST operate in a pass-by-value mode. A 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.

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

The DRMAA specification assumes that destination languages for a binding typically support the concept of exceptions. If a destination language does not support them (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.

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

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

Language bindings SHOULD define numerical values for all constants and enumeration members, in order to support binary portability of DRMAA-based applications.

The following table shows a non-normative example for a language binding definition:

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

The DRMAA specification defines some custom value types, in order to express a special value semantic not expressible in IDL: `OrderedStringList, StringList, Dictionary` and `TimeAmount`. The language binding MUST replace these type definitions with semantically equal reference or value types in the according language. This may include the creation of new complex language types for one or more of the above concepts.

Java binding example

IDL	Java
`StringList`	`java.util.Set`
`OrderedStringList`	`java.util.List`

`TimeAmount`	`long`
`Dictionary`	`java.util.Map`

Queues and slots

DRMAA abstracts different resource-related concepts of modern DRM systems. Two of them are queues and slots. Even though DRMAA has support for requesting queues and slots and monitoring them, it cannot assign any generalized semantic to these terms. This is reasoned by the different interpretation in different DRM systems. As one example, queues can be either treated as representation of execution hosts (Sun Grid Engine) or as central waiting line located at the scheduler (LSF). The support for queues and slot therefore needs to be understood as work with opaque information. The monitoring part just provides a non-exhaustive list of valid inputs for the attribute counterparts in the job template.

Lifetime of job information

The job monitoring model in DRMAA assumes that the DRM system has three job information states: running, buffered, purged. Only information for jobs that are still running or are still held in the buffer of finished job information will be reported. Jobs that have been purged out to accounting will be ignored. The lifetime of job information in the buffered state depends on the fact whether or not the according job was submitted in a DRMAA job session.

Changes in comparison to GFD.130

Changes in comparison to GFD.130 (DRMAA 1.0)

- The IDL module name was change to DRMAA2, in order to intentionally break backward compatibility of the interface.
- 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 `getState()` 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).
- The description of the `FAILED` state was extended to support a more specific differentiation between different job failure reasons. The new `subState` feature allows the DRMAA implementation to provide better information, if available. There was no portable way of standardizing extended failure information in a better way. (conf. call May 12th 2009, solves issue #5875)
- The `nativeSpecification` attribute in the job template was renamed to `nativeOptions` for better understanding. The description text is now also more specific. (F2F meeting July 2009)
- Version 2.0 of DRMAA supports restartable sessions by the newly introduced `SessionManager` interface. It allows creating multiple concurrent sessions for job submission (solves issue #2821), which can be restarted by their generated session name (solves issue #2820). `Session.init()` and `Session.exit()` functionalities are moved to the according session creation and closing routines. The descriptions were fixed accordingly (solves issue #2822). The `AlreadyActiveSession` error was removed. (F2F meeting July 2009)
- The `drmaaImplementation` attribute was removed, since it was redundant to the `drmsInfo` attribute. This one is now available in the new `SessionManager` interface. (F2F meeting July 2009)
- DRMAA2 replaces the identification of jobs by strings with `Job` objects. This enables a tighter integration of job meta-data and identity, for the price of reduced performance in (so far not existing) DRMAA RPC scenarios. The former DRMAA `control()` with the `JobControlAction` structure is now split up into dedicated functions (such as `hold()` and `release()`) on the `Job` object. The former `HoldInconsistentStateException, ReleaseInconsistentStateException, ResumeInconsistentStateException`, and `SuspendInconsistentStateException` from DRMAA v1.0 are now expressed as single `InconsistentStateException` with different meaning per dedicated function. String list for job identifiers are replaced by `Job` object lists (F2F meeting July 2009)
- The binding of job template attribute names and exception names to strings was removed from the main specification. Language bindings such as for the C programming languages have to define their own mapping. It is recommended to keep string identifiers from DRMAA 1.0 as far as possible.
- The original separation between `synchronize()` and `wait()` was replaced by a complete new synchronization semantic in the API. DRMAA2 has now two methods, `waitStarted()` and `waitTerminated()`. The first waits for any state that expresses that the job was started, the second for any terminal status. Both methods are available on session level (wait for any of the given jobs to start / end) or on single job level (solves issue #5880 and #2838). The function returns always a `Job` object, in order to allow chaining, e.g.

job.wait(JobStatus.RUNNING).hold(). The session-level functions implement the old DRMAA wait(SESSION_ANY) semantics.
The old synchronize() semantics are no longer directly supported - instead, the DRMAA application should use a looped Job.wait*() / JobSession.waitAny*() call. The result is a more condensed and responsive API, were the application can decide to keep the user informed during synchronization on a set of jobs. DRMAA library implementations should also become easier to design, since the danger of multithreading side effects inside the DRMAA API is reduced by this change.
As a side effect, JOB_IDS_SESSION_ANY and JOB_IDS_SESSION_ALL are no longer needed. The special consideration of a partial failures during SESSION_ALL wait activities is also no longer necessary (F2F meeting July 2009)
- Issue #5877 (support for direct job signaling) was rejected, even though there was an according request from the SAGA WG.
- Issue #2782 (change attributes of submitted, but pending jobs) was rejected.
- DRMAA2 supports the monitoring of execution resources through the MonitoringSession interface. The JobInfo value type was heavily extended for providing more information (solves issue #2827).
- The DRMAA JobSession interface can additionally support a callback facility, where the DRMAA library informs the application about state change events in the DRM system.
- The InconsistentStateException was removed, since it is semantically equal to the IllegalStateException (conf. call Jan 7th 2010)
- Event though the DRMAAv2 surveys showed interest in interactive job support, this feature was intentionally left out. Reasons are the missing support in some major DRM systems, and the lack of a relevant DRMAA-related use case (conf. call Jan 7th 2010)
- SAGA is realizing asynchronous operation based on the assumption that re-entrancy is supported by DRMAA implementations. For this reason, thread safety discussions are now part of every single method description.
- New job template attributes for file transfers were introduced. They allow to express a set of file staging activities, similar to the approach in LSF and SAGA. They replace the old transferFiles attribute, the according FileTransferMode data structure and the special host definition syntax in inputPath / outputPath / errorPath (different conf. calls, SAGA F2F meeting, solves issue #5876)
- The different suspend job states from DRMAA1 (user suspended, system suspended, user / system suspended) are now combined into one suspend state. DRM systems with the need to express the different suspend reasons can use the new sub-state feature (conf. call Mar 5th 2010).
- Resource usage limitations can now be expressed by two dictionaries - (softResourceLimits and hardResourceLimits) and an according standardized set of valid dictionary keys (LimitType). The idea is to allow a direct mapping to ulimit(3) semantics, which are supported by the majority of DRM system today. A separate run duration limit is no longer needed, since this is covered by the new CPU_TIME limit parameter. (conf. call Jun 9th 2010).
- JobInfo::hasCoreDump is no longer supported, since the information is useless without according core file staging support, which is not implementable in a portable way. (conf. call Jun 9th 2010)
- JobTemplate is now a value type, meaning that it maps to a struct in C. This removes the need for DRMAA-defined methods for construction and destruction of job templates. An eventual RPC scenario for DRMAA gets easier with this approach, since it is closer to the JSDL concept of a job description document. Supported string placeholders for job template attributes are now listed in the JobTemplatePlaceholder enumeration, and must be filled with values by the language binding. Invalid job template settings are now only detected on job submission, not when the attribute is set. Implementation-specific job template extensions are now covered by the generic drmsAttributes dictionary. This more generic approach also makes nativeOptions obsolete, so it was removed. Implementations should support all relevant native settings explicitly as keys in the drmsAttributes dictionary. (conf. call May 26th 2010).
- The JobSession now allows to fetch also information about jobs that were not submitted through DRMAA (conf. call June 23th 2010).
- The blockEmail attribute in the JobTemplate was removed. (conf. call July 28th 2010).

DRMAAv2 API

DRMAAv2 IDL Definition

The following text shows the complete IDL description of the DRMAA2 interface. Later chapters in this document explain the different parts.

```
module DRMAA2 {
```

See Data Types chapter for more information about the following part:

```
valuetype OrderedStringList sequence<string>;
valuetype StringList sequence<string>;
valuetype Dictionary sequence< sequence<string,2> >;
valuetype AbsoluteTime string;
valuetype TimeAmount long long;

const TimeAmount TIMEOUT_WAIT_FOREVER;
const TimeAmount TIMEOUT_NO_WAIT;

enum JobState {
 UNDETERMINED, QUEUED, QUEUED_HELD, RUNNING, SUSPENDED, REQUEUED, REQUEUED_HELD, DONE, FAILED
};

enum OperatingSystem {
 HPUX, LINUX, IRIX, TRUE64, MACOS,
 SUNOS, WIN, WINNT, AIX, UNIXWARE, BSD, OTHER
};

enum CpuArchitecture {
 ALPHA, ARM, CELL, PA-RISC, X86, X64, IA-64,
 MIPS, PPC, PPC64, SPARC, SPARC64, OTHER
};
```

See JobInfo chapter for more information about the following part:

```
struct JobInfo {
 readonly attribute string jobId;
 readonly attribute Dictionary resourceUsage;
 readonly attribute long exitStatus;
 readonly attribute string terminatingSignal;
 readonly attribute string annotation;
 readonly attribute JobState jobState;
 readonly attribute native jobSubState;
 readonly attribute OrderedStringList allocatedMachines;
 readonly attribute string submissionMachine;
 readonly attribute string jobOwner;
 readonly attribute string queueName;
 readonly attribute TimeAmount wallclockTime;
 readonly attribute long cpuTime;
 readonly attribute AbsoluteTime submissionTime;
 readonly attribute AbsoluteTime dispatchTime;
 readonly attribute AbsoluteTime finishTime;
};
```

See Exceptions chapter for more information about the following part:

```
exception AuthorizationException {string message;};
exception DefaultContactStringException {string message;};
exception DeniedByDrmException {string message;};
exception DrmCommunicationException {string message;};
exception TryLaterException {string message;};
exception SessionManagementException {string message;};
exception TimeoutException {string message;};
exception InternalException {string message;};
exception InvalidArgumentException {string message;};
exception InvalidSessionException {string message;};
exception InvalidStateException {string message;};
exception OutOfMemoryException {string message;};
       exception UnsupportedAttributeException {string message;};
       exception UnsupportedOperationException {string message;};
```

See SessionManager chapter for more information about the following part:

```
interface SessionManager{
 struct Version {
  readonly attribute long major;
  readonly attribute long minor;
 };

 readonly attribute string systemInfo;
 readonly attribute Version version;

 ReservationSession createReservationSession(in string sessionName,  in string contactString)
  raises (???);
 ReservationSession openReservationSession(in string sessionName)
  raises (???);
 void closeReservationSession(in ReservationSession s)
  raises (???);
 void destroyReservationSession(in string sessionName)
  raises (???);

 StringList getReservationSessions()
  raises (???);

 JobSession createJobSession(in string sessionName, in string contactString)
  raises (???);
 JobSession openJobSession(in string sessionName)
  raises (???);
 void closeJobSession(in JobSession s)
  raises (???);
 void destroyJobSession(in string sessionName)
  raises (???);

 StringList getJobSessions()
  raises (???);

 MonitoringSession createMonitoringSession (in string contactString)
  raises (???);
 void closeMonitoringSession(in MonitoringSession s)
  raises (???);
};
```

See DrmaaCallback chapter for more information about the following part:

```
enum DrmaaEvent {
 NEW_STATE_UNDETERMINED, NEW_STATE_QUEUED_ACTIVE,
 NEW_STATE_HOLD, NEW_STATE_RUNNING,
 NEW_STATE_SUSPENDED, NEW_STATE_DONE,
 NEW_STATE_FAILED, MIGRATED, ATTRIBUTE_CHANGE
};

struct DrmaaNotification {
 readonly attribute DrmaaEvent event;
 readonly attribute Job job;
};

interface DrmaaCallback {
 void notify(in DrmaaNotification notification)
};
```

See ReservationSession chapter for more information about the following part:

```
 interface ReservationSession {
  readonly attribute string contact;
  readonly attribute string sessionName;

  Reservation getReservation(string reservationId);

  Reservation requestReservation(in ReservationTemplate reservationTemplate)
   raises (???);

  sequence<Reservation> getReservations();
   raises (???);

 };
```

See ReservationTemplate chapter for more information about the following part:

```
 struct ReservationTemplate {
                readonly attribute StringList attributeNames;            // must be supported
attribute string reservationName;                        // must be supported
attribute string configurationName;                      // must be supported
attribute AbsoluteTime startTime;                        // must be supported
attribute AbsoluteTime endTime;                          // must be supported
attribute TimeAmount duration;                           // support is optional
attribute long minSlots;                                 // must be supported
attribute long maxSlots;                                 // must be supported
attribute StringList candidateMachines;                  // support is optional
attribute long minPhysMemory;                            // support is optional
attribute OperatingSystem machineOS;                     // support is optional
attribute CpuArchitecture machineArch;                   // support is optional
attribute Dictionary drmsSpecific;                       // must be supported
};
```

See ReservationInfo chapter for more information about the following part:

```
 struct ReservationInfo {
   readonly attribute sequence <string> reservedMachines;
   readonly attribute AbsoluteTime reservedStartTime;
   readonly attribute AbsoluteTime reservedEndTime;
   readonly attribute string comment;
                readonly attribute Dictionary drmsSpecific;
 };
```

See Reservation chapter for more information about the following part:

```
 interface Reservation {
   readonly attribute string reservationId;
         readonly attribute ReservationSession session;
  readonly attribute ReservationTemplate reservationTemplate;
  ReservationInfo getInfo()
   raises (???);
   void terminate(); // cancels a reservation
raises (???);

 };
```

See JobSession chapter for more information about the following part:

```
interface JobSession {
 readonly attribute string contact;
 readonly attribute string sessionName;
 sequence<Job> getJobs(JobInfo filter);

 Job runJob(in DRMAA::JobTemplate jobTemplate)
  raises (???);
 sequence<Job> runBulkJobs( in DRMAA::JobTemplate jobTemplate,
     in long beginIndex,
     in long endIndex,
     in long step)
  raises (???);
 Job waitAnyStarted(in sequence<Job> jobs, in TimeAmount timeout)
  raises (???);
 Job waitAnyTerminated(in sequence<Job> jobs, in TimeAmount timeout)
  raises (???);
 void registerEventNotification(in DrmaaCallback callback)
  raises (???);
       }
```

See JobTemplate chapter for more information about the following part:

```
 enum JobSubmissionState {
  HOLD_STATE, ACTIVE_STATE
 };

 enum ResourceLimitType {
  CORE_FILE_SIZE, DATA_SEG_SIZE, FILE_SIZE, OPEN_FILES, PIPE_SIZE,
  STACK_SIZE, CPU_TIME, MAX_USER_PROCESSES, VIRTUAL_MEMORY, WALLCLOCK_TIME
 };

 enum JobTemplatePlaceholder {
  HOME_DIRECTORY, WORKING_DIRECTORY, PARAMETRIC_INDEX
 };

 struct JobTemplate {
  readonly attribute StringList attributeNames;            // must be supported
attribute string remoteCommand;                          // must be supported
attribute OrderedStringList args;                        // must be supported
attribute DRMAA::JobSubmissionState jobSubmissionState;  // must be supported
attribute boolean rerunnable;                            // must be supported
attribute Dictionary jobEnvironment;                     // must be supported
attribute string workingDirectory;                       // must be supported
attribute string jobCategory;                            // must be supported
attribute string accountingId;                           // must be supported
attribute StringList email;                              // must be supported
attribute boolean emailOnStart;                          // support is optional
attribute boolean emailOnEnd;                            // support is optional
attribute string jobName;                                // must be supported
attribute string inputPath;                              // must be supported
attribute string outputPath;                             // must be supported
attribute string errorPath;                              // must be supported
attribute boolean joinFiles;                             // must be supported
attribute string reservationId;                          // must be supported
attribute string queueName;                              // must be supported
attribute long minSlots;                                 // must be supported
attribute long maxSlots;                                 // must be supported
attribute long priority;                                 // must be supported
attribute StringList candidateMachines;                  // must be supported
attribute long minPhysMemory;                            // must be supported
attribute OperatingSystem machineOS;                     // must be supported
attribute CpuArchitecture machineArch;                   // must be supported
attribute AbsoluteTime startTime;                        // must be supported
attribute AbsoluteTime deadlineTime;                     // support is optional
attribute Dictionary stageInFiles;                       // support is optional
attribute Dictionary stageOutFiles;                      // support is optional
attribute Dictionary softResourceLimits;                 // support is optional
attribute Dictionary hardResourceLimits;                 // support is optional
readonly attribute string accountingId;
// support is optional
attribute Dictionary drmsSpecific;                       // must be supported
}
```

Extend Placeholders
According to issue #2837, more placeholders should be supported. Was favored by most survey participants. Demands research about common placeholders in today's DRM systems. Issue #5873 also demands 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.

See Job chapter for more information about the following part:

```
 interface Job {
  readonly attribute string jobId;
  readonly attribute JobSession session;
  readonly attribute JobTemplate jobTemplate;
  readonly attribute Reservation reservation;
  void suspend()  // suspend a running job
raises (???);
  void resume()  // resume a suspended job
raises (???);
  void hold()   // put a queued job on hold
raises (???);
  void release() // release a job on hold
raises (???);
  void terminate()  // terminate a running job
raises (???);
  JobState getState(out native subState)
   raises (???);
  JobInfo getInfo()
   raises (???);
  Job waitStarted(in TimeAmount timeout)
   raises (???);
  Job waitTerminated(in TimeAmount timeout)
   raises (???);
 };
```

See MonitoringSession chapter for more information about the following parts:

```
   struct Queue {
      readonly attribute string name;
      readonly attribute TimeAmout maxWallclockTime;
   }
```

```
   struct Machine {
      readonly attribute string name;
      // number of processor sockets in the machine
readonly attribute long sockets;
      // number of CPU cores per socket, ASMP ?
readonly attribute long coresPerSocket;
      // number of hardware supported threads per CPU core
readonly attribute long threadsPerCore;
      // Load average on socket, average interval ?
readonly attribute double load;
      // Physical memory installed in the machine
readonly attribute long physicalMemory;
      // virtual memory available in the machine
readonly attribute long virtualMemory;
      // Operating system on the machine
readonly attribute OperatingSystem operatingSystem;
      // The version of the operating system on machine as string
readonly attribute String operatingSystemVersion;
      // Processor architecture
readonly attribute CpuArchitecture arch;
   }
```

```
 interface MonitoringSession {
   const long NO_LIMIT;

  //// attributes and methods on DRM system level ////
readonly attribute StringList drmsVersionString;
  sequence<Reservation> getAllReservations();
  sequence<Job> getAllJobs(JobInfo filter);
                sequence<Queue> getAllQueues(StringList names);
                sequence<Machine> getAllMachines(StringList names);
  readonly attribute StringList drmsJobCategoryNames;
 };
}
```

Data Types

Custom value types

The following value types must be provided by a language mapping of DRMAA, since they are used in different occasions through the API:

- `valuetype OrderedStringList sequence<string>`: An unbounded list of strings, which is ordered automatically.
- `valuetype StringList sequence<string>`: An unbounded unordered list of strings.
- `valuetype Dictionary sequence<sequence<string,2>>`: An unbounded dictionary type for storing key-value pairs.
- `valuetype AbsoluteTime string`: Expression of a point in time, at least with a resolution to seconds. A string representation according to RFC822 must be supported.
- `valuetype TimeAmount long long`: Expression of an amount of time, at least with a resolution to seconds. A string representation according to RFC822 must be supported.

Constant definitions

The following constants are possible arguments in two multiple interfaces, and must be therefore defined by the language binding on module scope:

- `TimeAmount TIMEOUT_WAIT_FOREVER`: Expresses an indefinite amount of time.
- `TimeAmount TIMEOUT_NO_WAIT`: Expresses an zero amount of time.

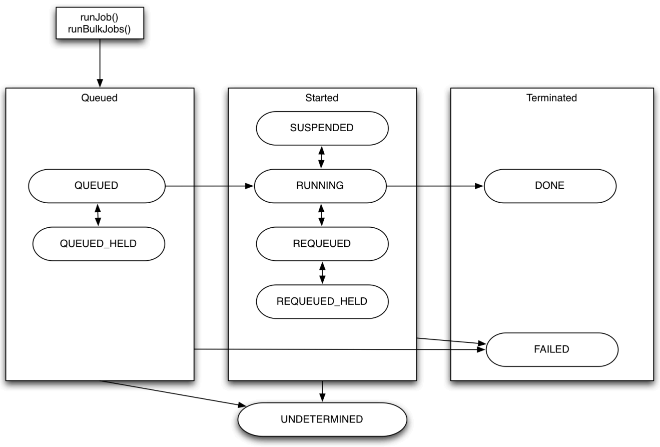
JobState

The `JobState` enumeration defines the possible return values when asking for the state of a job. The elements have the following meaning:

- `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`: The job is queued for being scheduled and executed.
- `JobState:QUEUED_HELD`: 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:SUSPENDED`: The job has been suspended by the user, the system or the administrator.
- `JobState:REQUEUED`: The job was re-queued by the DRM system.
- `JobState:REQUEUED_HELD`: The job was re-queued by the DRM system.
- `JobState:DONE`: The job finished without an error.
- `JobState:FAILED`: The job exited abnormally before finishing.

A DRMAA implementation is not required to possibly return all of the job states defined 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 as given here. An implementation SHOULD NOT return any job state value other than those defined in the `JobState` enumeration.

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



The figure expresses also the categorization of possible job states into 'Queued', 'Started', and 'Terminated'. This distinguishing is relevant for the different wait functions in DRMAA, which are related to one state category, instead of one particular state.

Implementations SHALL NOT introduce other job transitions or states beside the ones stated above. In case of additional job transitions (e.g. RUNNING -> QUEUED), implementations SHOULD emulate the neccessary intermediate steps. In case of additional job states, implementations SHOULD map them to the `subState` information available in the `Job::getState()` method. One example are sub-states for staged-in and staged-out.

The DRMAA job state model can be mapped to other high-level API state models:

TODO: Fix the table, according to latest state model.

DRMAA Job State	SAGA Job State [SAGA]	OGSA-BES Job State [OGSABES]
UNDETERMINED	N/A	N/A
QUEUED	Running	Pending (Queued)
HOLD	Running	Pending (Held)
RUNNING	Running	Running (Executing)
SUSPENDED	Suspended	Running (Suspended)
DONE	Done	Finished
FAILED	Cancelled, Failed	Cancelled, Failed

OperatingSystem

The `OperatingSystem` enumeration is used as data type both in the advanced reservation and the DRM system monitoring functionalities. It expresses as set of standardized identifiers for operating system types. The list is a shortened version of the according CIM Schema, with respect to operating systems that are supported by the majority of DRM systems available:

- `HPUX`: HP-UX Unix by Hewlett-Packard.
- `LINUX`: All operating system distributions based on the Linux kernel.
- `IRIX`: The IRIX operating system by SGI.
- `TRUE64`: True64 Unix by Hewlett-Packard, or DEC Digital Unix, or DEC OSF/1 AXP.
- `MACOS`: The MAC OS X operating system by Apple.
- `SUNOS`: SunOS resp. Solaris operating system by Sun.
- `WIN`: Microsoft Windows operating systems not based on the NT kernel: Windows 95, Windows 98, Windows ME.
- `WINNT`: Microsoft Windows operating systems based on the NT kernel
- `AIX`: AIX Unix by IBM.
- `UNIXWARE`: UnixWare system by SCO group.
- `BSD`: All operating system distributions based on the BSD kernel.
- `OTHER`: An operating system type not specified in this list.

The operating system information is only useful in conjunction with a version string, which models the reporting approach taken in most DRM systems. Examples:

- The Apple MacOS X operating system commonly denoted as 'Snow Leopard' would be reported as "MACOS" with the version string

"10.6".
- The Microsoft Windows 7 operating system would be reported as "WINNT" with the version information "6.1", which is the internal version number reported by the Windows API.
- All Linux distributions would be reported as operating system type "LINUX" with the major revision of the kernel, e.g. "2.6"
- The Solaris operating system is reported as "SUNOS", together with the internal version number, e.g. "5.10" for Solaris 10.

The DRMAA `OperatingSystem` enumeration can be mapped to the according data model in JSDL:

DRMAA OperatingSystem value	JSDL jsdl:OperatingSystemTypeEnumeration value [JSDL]
HPUX	HPUX
LINUX	LINUX
IRIX	IRIX
TRUE64	Tru64_UNIX / OSF
MACOS	MACOS
SUNOS	SunOS / SOLARIS
WIN	WIN95 / WIN98 / Windows_R_Me
WINNT	WINNT / Windows_2000 / Windows_XP
AIX	AIX
UNIXWARE	SCO_UnixWare / SCO_OpenServer
BSD	BSDUNIX / FreeBSD / NetBSD / OpenBSD
OTHER	Other

CpuArchitecture

The `CpuArchitecture` enumeration is used as data type both in the advanced reservation and the DRM system monitoring functionalities. It expresses as set of standardized identifiers for processor architectures. The list is a shortened version of the according CIM Schema, with respect to processor architectures that are supported by available DRM systems:

- `ALPHA`: The DEC Alpha / Alpha AXP processor architecture.
- `ARM`: The ARM processor architecture.
- `CELL`: The Cell processor architecture.
- `PA-RISC`: The PA-RISC processor architecture.
- `X86`: The IA-32 line of the X86 processor architecture family, with 32bit support.
- `X64`: The X86-64 line of the X86 processor architecture family, with 64bit support.
- `IA-64`: The Itanium processor architecture.
- `MIPS`: The MIPS processor architecture.
- `PPC`: The PowerPC processor architecture, all models with 32bit support only.
- `PPC64`: The PowerPC processor architecture, all models with 64bit support.
- `SPARC`: The SPARC processor architecture, all models with 32bit support only.
- `SPARC64`: The SPARC processor architecture, all models with 64bit support.
- `OTHER`: A processor architecture not specified in this list.

The DRMAA `CpuArchitecture` enumeration can be mapped to the according data model in JSDL:

DRMAA CpuArchitecture value	JSDL jsdl:ProcessorArchitectureEnumeration value [JSDL]
ALPHA	other
ARM	arm
CELL	other
PA-RISC	parisc
X86	x86_32

X64	x86_64
IA-64	ia64
MIPS	mips
PPC	powerpc
PPC64	powerpc
SPARC	sparc
SPARC64	sparc
OTHER	other

DrmaaCallback

DrmaaEvent enumeration

- `NEW_STATE_UNDETERMINED`:
- `NEW_STATE_QUEUED_ACTIVE`:
- `NEW_STATE_HOLD`:
- `NEW_STATE_RUNNING`:
- `NEW_STATE_SUSPENDED`:
- `NEW_STATE_DONE`:
- `NEW_STATE_FAILED`:
- `MIGRATED`: The job was migrated to a new machine and is running again.
- `ATTRIBUTE_CHANGED`: A monitoring attribute of the job, such as the memory consumption, changed to a new value.

Exceptions

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.

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 function. Such a language binding SHOULD specify numeric values for all DRMAA error constants.

- `AuthorizationException` - The user is not authorized to perform the given function.
- `DefaultContactStringException` - The DRMAA implementation could not use the default contact string to connect to DRM system.
- `DeniedByDrmException` - The DRM system rejected the job. The job is not accepted due to current DRM configuration settings conflicting with the job template settings, or because of syntactical issues with the job template attributes (e.g. invalid URL's or host names).
- `DrmCommunicationException` - Could not contact DRM system. The problem source is unknown to the library implementation, so it is unknown if the problem is transient or not.
- `TryLaterException` - The DRMAA implementation detected a transient problem with performing the function, for example due to excessive load. The application is recommended to retry the call.

- `SessionManagementException` - A problem was encountered while trying to create / open / close / destroy a session.
- `TimeoutException` - The timeout given in one the job waiting functions was reached without finishing the function.
- `InternalException` - An unexpected or internal error occurred in the DRMAA library, for example a system call failure.
- `InvalidArgumentException` - From the viewpoint of the DRMAA library, a function parameter is invalid or inappropriate for the particular function call.
- `InvalidSessionException` - The session used for the function is not valid, for example since it was closed before.
- `InvalidStateException` - The function call is not allowed in the current state of the job.
- `OutOfMemoryException` - This exception can be thrown by any method at any time when the DRMAA implementation has run out of free memory.
- `UnsupportedAttributeException` - The optional attribute is not supported by the DRMAA implementation. This exception may either be raised by the setter function for the attribute or by the job submission function, depending on the semantics defined by the language binding.
- `UnsupportedOperationException` - The function is not supported by the DRMAA implementation. One example is the registration of an event callback function.

Job

Job interface

The following chapter explains the set of constants, methods and attributes defined in the `Job` interface. Every job in the session is expressed by an own instance of the `Job` interface. It allows to instruct the DRM system for a job status change, and to query the status attributes of the job in the DRM system. Status values read from the `Job` object SHOULD reflect the current status of the job in the DRM system at the time of the call.

The job control functions allow modifying the status of the single job in the DRM system. They 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. The behavior is implementation-specific.

Thread Safety
To avoid thread races in multi-threaded applications, the DRMAA implementation user should explicitly synchronize this call with any other call to the same object. This MAY be already realized by the DRMAA implementation.

suspend

void suspend()

Thread Safety

resume

void resume()

Thread Safety

hold

void hold()

Thread Safety

release

void release()

Thread Safety

terminate

void terminate()

Thread Safety

waitStarted

Job waitStarted(in TimeAmount timeout)

Thread Safety

waitTerminated

Job waitTerminated(in TimeAmount timeout)

Thread Safety

getState

JobState getState(out native subState)

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. It is up to the implementation to determine whether this method is capable of operating on jobs submitted outside of the current DRMAA session.

The `getState` 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.

Thread Safety

Issue #2824 – Clarify status query on reaped jobs

getInfo

JobInfo getInfo()

Thread Safety

JobInfo

The `JobInfo` structure describes job information that is available to a DRMAA based application. The structure is used in two occasions - first for the expression of one job status, and second as filter expression when retrieving a list of jobs from the DRMAA library.

In both usage scenarios, the structure information has to be understood as snapshot of the live DRM system. Multiple values being set in one structure instance should be interpreted as 'occurring at the same time'. In real implementations, some granularity limits must be assumed - for example, the wallclockTime and the cpuTime attributes might hold values that were measured with a very small delay one after each other.

In the use case of job information monitoring, it is assumed that the DRM system has three job information states: running, buffered, purged. Only information for jobs that are still running or are still held in the buffer of finished job information will be reported completely. If jobs have been purged out to accounting, different attributes might not contain valid data. Implementations MAY decide to return only partially filled `JobInfo` instances due to performance restrictions in the communication with the DRM system.

A language binding MUST provide an according mechanism to identify that a member has an invalid value (UNSET).

readonly attribute string jobId;

For monitoring: Returns the string job identifier assigned to the job by the DRMS.

For filtering: Returns the job with the chosen job identifier.

readonly attribute Dictionary resourceUsage;

For monitoring: Returns resource consumption informstion for the given job. The dictionary keys are implementation-specific.

For filtering: Returns the jobs that have the dictionary key-value pairs as subset of their own.

readonly attribute long exitStatus;

For monitoring: The process exit status of the job, as reported by the operating system. If the job is not in one of the terminated states, the value should be UNSET.

For filtering: Return the jobs with the given exitStatus value. Jobs without exitStatus should be filtered out by asking for the appropriate states.

readonly attribute string terminatingSignal;

For monitoring: This attribute specifies the UNIX signal that reasoned the ending of the job. Implementations should document the extent to which they can gather such information in the particular DRM system (e.g. with Windows hosts).

For filtering: Returns the jobs with the given terminatingSignal value.

readonly attribute string annotation;

For monitoring: Gives a human-readable annotation describing why the job is in its current state or substate. The support for this information is optional.

For filtering: This attribute is ignored for filtering.

readonly attribute JobState jobState;

For monitoring: This attribute specifies the job's current state according to the DRMAA job state model.

For filtering: Returns all jobs in the specified state.

readonly attribute native jobSubState;

Every job state information in DRMAA can be extended by a `jobSubState` property, which expresses implementation- and DRMS-specific information about the particular state. The value of this attribute is implementation-specific, but should be documented accordingly.

For monitoring: This attribute specifies the job's current DRMAA implementation specific substate

For filtering: Returns all jobs in the specified substate.

readonly attribute OrderedStringList allocatedMachines;

This attribute expressed the set of machines that are utilized for job execution. Implementations MAY decide to give the ordering of machine names a particular meaning, for example putting the master node in a parallel job at first position. This decision should be documented for the user.

For monitoring: This attribute lists the set of names of the machines to which this job has been assigned.

For filtering: Returns the list of machines which have a set of assigned machines that is a superset of the given set of machines.

readonly attribute string submissionMachine;

For monitoring: This attribute specifies the machine from which this job was submitted.

For filtering: Returns the set of jobs that were submitted from the specified machine.

readonly attribute string jobOwner;

For monitoring: This attribute specifies the job owner as reported by the DRM system.

For filtering: Returns all jobs owned by the specified user.

readonly attribute string queueName;

For monitoring: This attribute specifies the queue in which the job was queued/started. As queues are a somewhat abstract concept, this attribute is highly DRM system dependent.

For filtering: Returns all jobs that are/were queued/started in the specified queue.

readonly attribute TimeAmount wallclockTime;

For monitoring: Time the job was running, including time the job was suspended. Equals to (finish time | now) - dispatch time.

For filtering: Returns all jobs that have consumed at least the specified amount of wall clock time.

readonly attribute long cpuTime;

For monitoring: This attribute specifies the amount of CPU time consumed by the specified job. This time does not include time spent in a suspended, requeued, or requeued held state.

For filtering: Returns all jobs that have consumed at least the specified amount of CPU time.

readonly attribute AbsoluteTime submissionTime;

For monitoring: This attribute specifies the time at which the job was submitted, according to the DRM system.

For filtering: Returns all jobs that were submitted at or after the specified submission time.

readonly attribute AbsoluteTime dispatchTime;

For monitoring: The time the job first entered a started state. For restarted or rescheduled jobs, this value will not change.

For filtering: Returns all jobs that entered a started state at or after the specified dispatch time.

readonly attribute AbsoluteTime finishTime;

For monitoring: The time the job first entered a terminated state.

For filtering: Returns all jobs that entered a terminated state at or after the specified finish time.

For additional attributes, such as DRMS-specific attributes, the JobInfo struct should be extended with a library-specific implementation. The mechanism for specifying the extended JobInfo struct is to be defined by the specific language binding, but the extended attributes offered are the domain of the specific library implementation.

JobSession

JobSession interface

Job sessions in the DRMAA API are intended to control the job submission and control activities. Every DRMAA `JobSession` object provides a container for the jobs submitted by its `runJob` resp. `runBulkJobs` methods. They are persisted by the library implementation, as described in the SessionManager chapter.

The interface provides one read-only attribute contact. It contains

contact

readonly attribute string contact

The attribute provides the contact string used on creation of this `JobSession` instance, or the default contact string if none was chosen.

sessionName

readonly attribute string sessionName

The attribute provides the session name used on creation or re-opening of this `JobSession` instance.

getJobs

sequence<Job> getJobs(JobInfo filter);

The function returns the list of job objects that match to the particular filter structure. The usage of `JobInfo` as a filter expression is described in the JobInfo chapter. If the filter argument is null, all jobs that are known to belong to that session are returned.

runJob

Job runJob(in DRMAA::JobTemplate jobTemplate) raises (???)

The `runJob` method SHALL submit a job with attributes defined in the job template given as a parameter. It returns a `Job` object that represents the job in the underlying DRM system.

> ⚠ runJob / runBulkJobs method
> Issue #5884 – apply solution from GFD.133 here.

runBulkJobs

sequence<Job> runBulkJobs(in DRMAA::JobTemplate jobTemplate, in long beginIndex, in long endIndex, in long step) raises (???)

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 `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`.

The `runBulkJobs` method SHOULD return a list of `Job` objects, were each of them represents a job in the underlying DRM system.

> ℹ Thread Safety

waitAnyStarted

Job waitAnyStarted(in sequence<Job> jobs, in TimeAmount timeout) raises (???)

This method SHALL wait for any of the specified jobs to enter one of the “started” states (see Section DataTypes). If the provided list contains jobs that are not part of the session, the call to `waitAnyStarted` SHALL fail, throwing 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 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.

This method SHALL return the `Job` object for the job that reached one of the “started” states.

Thread Safety
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.

This puts a lot of burden onto the DRMAA user, which might be no longer needed with the new wait() semantics.

In a multi-threaded environment with multiple `waitAnyStarted` calls, only one of the active thread gets the status change notification for a particular job, while the other threads continue waiting. If there are no more queryable jobs left in the session, all remaining waiting threads SHOULD fail with an `InvalidJobException`.
If thread A is waiting for a specific job with `Job.wait()`, and another thread, thread B, waiting for that same job or with `JobSession.waitAnyStarted()`, receives notification that the job has finished, thread A SHOULD fail with an `InvalidJobException`.

issue #5879
Solution applied to GFD.133 needs to be reflected also here.

waitAnyTerminated

Job waitAnyTerminated(in sequence<Job> jobs, in TimeAmount timeout) raises (???)

Thread Safety

Make clear that user should perform looped calls of this function, by reducing the list of checked jobs with every returning result.
There is also no reaping here, which is not obvious.

registerEventNotification

void registerEventNotification(in DrmaaCallback callback) raises (???)

Thread Safety

JobTemplate

JobTemplate structure

In order to define the attributes associated with a job, a DRMAA application uses the `JobTemplate` structure. A DRMAA application specifies in the template any required job parameters, and then passes the template to the DRMAA `JobSession` instance when requesting that a job be executed.

Overview

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. Optional attributes might or might not be supported by the library implementation. In the latter case, the job submission should fail for the according job template with a `UnsupportedAttributeException`.

A language binding specification SHOULD ensure that a `JobTemplate` instance is 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.

A language binding must define the notion of an "UNSET" value per data type used in the JobTemplate structure. Implementations MUST then set all attributes to this default value on struct allocation. This ensures that both the DRMAA application and the library implementation can determine untouched attribute members.

Phrase boolean in the negative

For languages which may not provide an inherit notion of struct introspection, the language binding SHOULD provide a read-only attributeNames attributes of type StringList that explicitly enumerates the 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 useful with the provided functions for setting and getting the values of the enumerated attributes.

Accessing implementation-specific attributes

A language binding MUST provide a means for accessing implementation-specific job template attributes, typically as additional members of the JobTemplate data structure.

Constants

The `JobTemplate` interface defines a set of constants that are used in the context of some of the attributes.

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.

JobSubmissionState

The `JobSubmissionState` enumeration is used as the type of the `JobTemplate::jobSubmissionState` interface attribute with 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.

Attributes

remoteCommand

attribute string 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.

args

attribute OrderedStringList args

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

jobSubmissionState

attribute DRMAA::JobSubmissionState jobSubmissionState

Defines the state of the job at submission time.

rerunnable

attribute boolean rerunnable

This flag indicates if the submitted job can safely be restarted by the DRM system, for example on a node failure or some other re-scheduling event. If the attribute is not set, the behavior is implementation-specific.

jobEnvironment

attribute Dictionary 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.

workingDirectory

attribute string 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` attribute 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.FAILED`.

jobCategory

DRMAA facilitates writing DRM-enabled applications even though the deployment properties, in particular the configuration of the DRMS, cannot be known in advance. Through the jobCategory string attribute, a DRMAA application can specify additional job needs that are to be mapped by the DRMAA implementation or DRM system to DRMS-specific options. It is intended as non-programmatic extension of DRMAA job submission capabilities. The interpretation of the `jobCategory` job template string attribute is implementation-specific, meaning that a DRMAA implementation could even map any or all category names to nothing. The order of precedence for DRMS configuration options produced by the `jobCategory` string attribute mapping versus those injected by the `drmsAttributes` dictionary is unspecified.

The value of the attribute SHOULD be one of the returned strings in `MonitoringSession::drmsJobCategoryNames`, otherwise an `InvalidAttributeValueException` should be thrown.

drmsSpecific

attribute Dictionary drmsSpecific

This dictionary attribute allows the DRMAA library user to pass DRMS-specific native options as key-value pairs during job submission. In contrast to the usage of predefined configuration sets with the `jobCategory` attribute, this allows to pass directly DRMS-specific options. As far as the DRMAA interface specification is concerned, the `drmsSpecific` dictionary keys are interpreted by each DRMAA library in its specific way.

One MAY use DRM configuration facilities, the `jobCategory` attribute or the `drmsSpecific` attribute at the same time. In this case, the `drmsSpecific` attribute SHOULD ultimately overrule other, even conflicting, configurations. Care SHOULD be exercised to not change the job submission call semantics, pass options that conflict the already set attributes, or violate the DRMAA API in any way. Implementations are free to reject job templates that have invalid or conflicting settings in this attribute.

email

attribute StringList email

This attribute holds a list of email addresses that is used to report DRM notifications. Content and formatting of the emails are defined by the respective DRM systems. If the list is empty or not set in the job template, no emails SHOULD be sent to the user running the job(s), even if the DRM system default behavior is to send emails on some event.

emailOnStart / emailOnEnd

attribute boolean emailOnStart / emailOnEnd

This flag indicates if the given email address should get a notification when the job was started resp. ended, according to the DRMAA state classes. If the email attribute is not set, this flag has no effect. With a given email address and no usage of this attribute in a job template, the behavior is implementation-specific.

jobName

attribute string 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.

inputPath / outputPath / errorPath

attribute string inputPath / outputPath / errorPath

Specifies the job's standard input / output / error stream 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 / output / error stream, unless the named configuration, native options, or a DRMS setting causes a source / destination for the stream to be set. If this attribute is set, it specifies a path in the local file system, which might also represent a mounted network file system. The parameter MUST be specified in a syntax that is common at the host where the job is executed, especially with respect to operating system differences (e.g. Unix vs. Windows).

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 to the job's user's home directory at the execution host.

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 execution host.

If set, and the job were successfully submitted, and the file can't be read / written before execution of the job, it enters the state `JobState.FAILED`.

If the `outputPath` or `errorPath` file does not exist at the time the job is about to be executed, the file SHALL first be created. An existing `outputPath` or `errorPath` file SHALL be opened in append mode.

joinFiles

attribute boolean 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`.

reservationId

attribute int reservationId

Specifies the id of the advance reservation associated with this job.

queueName

attribute string queueName

This attribute specifies the name of the queue the job should be submitted to. In case an empty string is provided, the implementation should use the DRM systems default queue, if existing. If no default queue is defined, or the given queue name is not valid, the implementation MUST throw an `InvalidAttributeValueException`.

The `MonitoringSession` interface supports the determination of valid queue names. If not restricted by other conditions in the job template, implementations SHOULD allow these queue names to be used in the `queueName` attribute. Implementations MAY also support other queue names.

Requesting a number of slots for a job in one queue has no implication on the number of utilized machines at run-time.

minSlots

attribute long minSlots

Minimum number of requested slots. The interpretation of the term 'slot' is DRM-specific. Implementations MAY interprete the slot count as number of concurrent processes being allowed on one machine. If this attribute is not specified, it should default to 1. If this attribute is greater than 1, the `jobCategory` MUST also be set, in order to express the nature of the intended parallel job execution.

maxSlots

attribute long maxSlots

Maximum number of requested slots. If this attribute is not specified, it should default to the value of `minSlots` attribute, or to 1 if the `minSlots` attribute is not specified.

candidateMachines

attribute StringList candidateMachines

Requests that the job should run on any subset, or all, of the given machines. If this attribute is not specified, it should default to the result of `MonitoringSession::drmsMachineNames`. If this resource demand cannot be fulfilled, an `InvalidAttributeValueException` should be thrown.

minPhysMemory

attribute long minPhysMemory

This attribute denotes the minimum amount of physical memory expected on the execution host(s). If this resource demand cannot be fulfilled, an `InvalidAttributeValueException` should be thrown.

machineOS

attribute OperatingSystem machineOS

Describes the expected operating system type on the execution host. If this resource demand cannot be fulfilled, an `InvalidAttributeValueException` should be thrown.

machineArch

attribute CpuArchitecture machineArch

Describes the expected machine architecture on the execution host. If this resource demand cannot be fulfilled, an `InvalidAttributeValueException` should be thrown.

startTime

attribute AbsoluteTime 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.

deadlineTime

attribute AbsoluteTime deadlineTime

Specifies a deadline after which the DRMS will abort 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.

fileTransfers

attribute Dictionary stageInFiles

attribute Dictionary stageOutFiles

Specifies what files should be transfered (staged) as part of the job execution. The Dictionary keys contain the source URLs from which data should be copy. The Dictionary values contain the destination URLs to which the data at the corresponding source URL (key) should be copied.

The data staging operation is defined to always be a copy operation.

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

softResourceLimits

attribute Dictionary softResourceLimits

This attribute specifies the soft limits on resource utilization of the job on the execution host. The valid dictionary keys are defined in the `LimitType` enumeration. An implementation is expected to map the setting to an `ulimit(3)` call in the operating system.

This attribute is optional. In case an implementation MUST throw an `UnsupportedAttributeException` if this attribute is not supported. If the attribute is supported, then all keys from `LimitType` must be accepted as valid setting.

hardResourceLimits

attribute Dictionary hardResourceLimits

This attribute specifies the hard limits on resource utilization of the job on the execution host. The valid dictionary keys are defined in the `LimitType` enumeration. An implementation is expected to map the setting to an `ulimit(3)` call in the operating system.

This attribute is optional. In case an implementation MUST throw an `UnsupportedAttributeException` if this attribute is not supported. If the attribute is supported, then all keys from `LimitType` must be accepted as valid setting.

accountingId

attribute string accountingId

This attribute denotes a string that can be used by the DRM system for job accounting purposes. Implementations SHOULD NOT utilize this information as authentication token, but only as additional identification information beside the library-dependent implicit security checks (see XXX).

priority

attribute long priority

This attribute specifies the scheduling priority for the job. The scheduling priority is to be interpreted by the DRMAA implementation or the DRM system in a DRM-specific fashion.

JSDL Mapping

Job Template Attribute	JSDL Attribute
jobName	<jsdl:jobName>
remoteCommand	<jsdl-hpcpa:Executable>
args	<jsdl-hpcpa:Argument>*
jobEnvironment	<jsdl-hpcpa:Environment>*
workingDirectory	<jsdl-hpcpa:WorkingDirectory>
inputPath	<jsdl-hpcpa:Input>
outputPath	<jsdl-hpcpa:Output>
errorPath	<jsdl-hpcpa:Error>

MonitoringSession

MonitoringSession interface

The `MonitoringSession` interface in DRMAA supports the monitoring of execution resources in the DRM system. This is distinct from the monitoring of jobs running in the DRM system, which is covered by the `JobSession` with the `Job` and the `JobInfo` interfaces.

The `MonitoringSession` interface supports four basic units of monitoring:

- Properties of the DRM system as a whole (e.g. DRM system version number) that are independent from the particular session resp. contact string
- Properties of the DRM system that depend on the current contact string (e.g. list of machines in the currently accessed Oracle Grid Engine cell)
- Properties of individual queues known from the `drmQueueNames` attribute
- Properties of individual machines available with the current contact string (e.g. amount of physical memory in a chosen machine)

All returned data is related to the current user running the DRMAA-based application. For example, the `getAllQueues` function, which MAY be reduced to only denote queues that are usable or generally accessible for the DRMAA application and user performing the query. Because no guarantee can be made as to future accessibility, and because of cases where list reduction may demand excessive implementation overhead in the DRMMA library, an unreduced or partially reduced list may be returned. The behavior of the DRMAA implementation in this regard SHOULD be clearly documented. In all cases, the list items MUST all be valid input for job submission via a JobTemplate.

drmsJobCategoryNames

```
readonly attribute StringList drmsJobCategoryNames
```

Contains the list of of valid job categories in this DRM system installation. Each category expresses a particular type of job execution that demands site-specific configuration, for example path settings, environment variables, or application starters such as MPIRUN. The possible return values SHOULD be coherent with the list published under:

http://www.drmaa.org/jobcategories/

In case the name is not taken from the DRMAA working group recommendations, it should be self-explanatory for the user to understand the implications on job execution. Implementations are recommended to provide a library configuration facility, which allows site administrators to link job category names with specific product- and site-specific configuration options, such as submission wrapper shell scripts.

drmsVersion

```
readonly atttribute version drmsVersion
```

This attribute denotes the DRM system version to which the current `MonitoringSession` belongs, as reported by the DRM system itself.

getAllMachines

```
sequence<Machine> getAllMachines()
```

This function returns the list of machines available in the DRM system. The returned list might be empty. The returned list might also contain hosts that are not accessible by the user (because of host configuration limits). The DRMS or the DRMAA implementation MAY reduce the list, however, based on machine or system policies.

getAllQueues

```
sequence<Queue> getAllQueues()
```

This function returns a list of supported queues for job submission. All queues named in this list SHOULD be usable in a job submission through DRMAA. The list can be empty and might not be complete. It might also contain queues that are not accessible by the user (because of queue configuration limits). The DRMS or the DRMAA implementation MAY reduce the list, however, based on queue, host, or system policies.

getAllJobs

```
sequence<Job> getAllJobs(JobInfo filter)
```

This function returns the list of all DRMS jobs visible to the user running the DRMAA-based application. In contrast to a `JobSession`, this result also contains jobs that were submitted outside of DRMAA (e.g. through command-line tools) by this user. The returned list MAY also contain jobs that were submitted by other users if the security policies of the DRM system allow such global visibility. The DRMS or the DRMAA implementation is at liberty, however, to restrict the set of returned jobs based on site or system policies, such as security settings or scheduler load restrictions.

Querying the DRM system for all jobs might result in returning an excessive number of Job objects. Implications to the library implementation are out of scope for this specification. Language bindings SHOULD NOT try to solve these scalability issues globally by replacing the `sequence` type of the return value with some iterator-alike solution. This approach would break the basic snapshot semantic intended for this method.

Machine struct

The `Machine` struct is used by the `MonitoringSession` to return information about an execution resource. The `Machine` struct contains read-only information only.

name

read-only attribute string name

This attribute contains the name of the machine as reported by the DRM system. The format of the machine name is implementation-specific, but MIGHT be a DNS host name. The naming scheme SHOULD be consistent for all strings returned.

operatingSystem

read-only attribute OperatingSystem operatingSystem

This attribute contains the operating system of the machine. Valid operating system names are specified in the `OperatingSystem` enumeration (see Data Types#OperatingSystem).

operatingSystemVersion

read-only attribute String operatingSystemVersion

This attribute contains the operating system version of the machine.

arch

read-only attribute CpuArchitecture arch

This attribute contains the instruction set architecture (ISA) of the machine. Valid instruction set architectures are specified in the CpuArchitecture enumeration (see Data Types#CpuArchitecture).

sockets

read-only attribute long sockets

This attribute contains the number of sockets available on the machine. The attribute value must be greater than 0. In the case where the correct value is unknown to the DRMAA implementation, the value must be set to 1.

coresPerSocket

read-only attribute long coresPerSocket

This attribute contains the number of cores offered per socket on the machine. The attribute value must be greater than 0. In case where the correct value is unknown to the DRMAA implementation, the value must be set to 1.

threadsPerCore

read-only attribute long threadsPerCore

This attribute contains the number of threads per execution core on the machine. The attribute value must be greater than 0. In case where the correct value is unknown to the DRMAA implementation, the value must be set to 1.

physicalMemory

read-only attribute long physicalMemory

This attribute contains the amount of physical memory in kilobytes available on the machine.

virtualMemory

read-only attribute long virtualMemory

This attribute contains the amount of virtual memory in kilobytes available on the machine.

load

read-only attribute double load

This attributes describes the 1-minute average load on the given machine, similar to the Unix `uptime` command. The value has only informative character, and should not be utilized by end user applications for job scheduling purposes. An implementation MAY provide delayed or averaged data here, if necessary due to implementation issues.

Queue struct

The `Queue` struct is used by the `MonitoringSession` to return information about a queue in the DRM system. No implication is made about the nature of definition of a queue in any DRM system, as the concept varies widely amount the various DRM system available in the market. The `Queue` struct contains read-only information only.

name

read-only attribute string name

This attribute contains the name of the queue as reported by the DRM system. The format of the queue name is implementation-specific. The naming scheme SHOULD be consistent for all strings returned.

maxWallclockTime

read-only attribute TimeAmount maxWallclockTime

This attribute contains the maximum amount of wallclock time allowed for jobs submitted to the queue. The attribute value is either a non-zero TimeAmount value, or NO_LIMIT when there is no restriction.

Reservation

Reservation interface

The following chapter explains the set of constants, methods and attributes defined in the `Reservation` interface.

> Thread Safety
> To avoid thread races in multi-threaded applications, the DRMAA implementation user should explicitly synchronize this call with any other call to the same object. This MAY be already realized by the DRMAA implementation.

reservationId

readonly attribute string reservationId

get reservation info

struct ReservationInfo getInfo() raises (???)

terminate

void terminate()

> Thread Safety

ReservationInfo

The ReservationInfo structure describes reservation information that is available to a DRMAA based application.

Thread Safety

reservedMachines

readonly attribute sequence <string> reservedMachines

Thread Safety

reservedStartTime

readonly attribute AbsoluteTime reservedStartTime

Thread Safety

reservedEndTime

readonly attribute AbsoluteTime reservedEndTime

Thread Safety

comment

readonly attribute string comment

Thread Safety

DRMS specific monitoring attributes - an extension point

readonly attribute Dictionary drmsSpecific

ReservationSession

ReservationSession interface

The following chapter explains the set of constants, methods and attributes defined in the `ReservationSession` interface. Every DRMAA `ReservationSession` object provides a container for the advance reservations created in the DRM system.

An application can open more than one reservation session at a time, but every reservation can only belong to one session. Sessions (in terms of their reservation list) should be persisted after closing the session by the `SessionManager.closeReservationSession` method, until they are explicitly reaped by the `SessionManager.destroyReservationSession` method.

The `ReservationSession` interface has explicit methods for creating and destroying reservation 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.

getReservations

sequence<Reservation> getReservations()

Thread Safety

createReservationTemplate

ReservationTemplate createReservationTemplate() raises (???)

The `createReservationTemplate` method SHALL return a new `ReservationTemplate` instance. The template is used to set the defining characteristics for advance reservations to be requested. Once the template has been created, it should also be deleted (via `deleteReservationTemplate` method) when no longer needed. Failure to do so may result in a memory leak.

> Thread Safety

deleteReservationTemplate

void deleteReservationTemplate(in ReservationTemplate reservationTemplate) raises (???)

The `deleteReservationTemplate` method is used to deallocate a reservation template, and SHALL perform all necessary steps required to free all memory associated with the given `ReservationTemplate` 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 existing advance reservations. This method MUST only work on `ReservationTemplate` instances that were created with the `createReservationTemplate` method and have not previously been deleted with the `deleteReservationTemplate` method and MUST otherwise throw an `InvalidReservationTemplateException`.

> Thread Safety

requestReservation

Reservation requestReservation(in ReservationTemplate reservationTemplate) raises (???)

The `requestReservation` method SHALL request an advance reservation in the DRM system with attributes defined in the reservation template given as a parameter. This method MUST only work on `ReservationTemplate` instances that were created with the `createReservationTemplate` method and have not previously been deleted with the `deleteReservationTemplate` method and MUST otherwise throw an `InvalidReservationTemplateException`.

The `requestReservation` method SHOULD return a `Reservation` object that represents the advance reservation in the underlying DRM system.

In case some of the reservation conditions are not fulfilled later (e.g. failing reservedHosts), it is assumed that the reservation remains valid.

> Thread Safety

ReservationTemplate

ReservationTemplate interface

In order to define the attributes associated with an advance reservation, a DRMAA application uses the `ReservationTemplate` interface. Instances of such templates are created via the active `ReservationSession` instance. A DRMAA application gets a `ReservationTemplate` from the active `ReservationSession` instance, specifies in the template any required advance reservation parameters, and then passes the template back to the DRMAA `ReservationSession` instance. When finished, the DRMAA application SHOULD call the `ReservationSession::deleteReservationTemplate` method to allow the underlying implementation to free any resources bound to the template instance.

Interface overview

A language binding specification MUST model the `ReservationTemplate` interface the same way as the `JobTemplate` interface is realized, with respect to the choosen getter / setter approach and the access to implementation-specific attributes. See section JobTemplate for a detailed description.

startTime

attribute AbsoluteTime startTime

Start of the time window in which resources should be reserved.

endTime

attribute AbsoluteTime endTime

End of the time window in which resources should be reserved.

duration

attribute TimeAmount duration

Reservation duration. If reservation duration is shorter than `endTime` - `startTime` the earliest reservation (matching the requirements) will be created. If this attribute is omitted then the duration is assumed to be equal to `endTime` - `startTime`.

reservationName

attribute string reservationName

Human-readable reservation name. If this attribute is omitted then the name of the reservation is implementation-defined.

minSlots

attribute long minSlots

Minimum number of requested slots. The interpretation of the term 'slot' is DRM-specific. Implementations MAY interprete the slot count as number of concurrent processes being allowed on one machine. If this attribute is not specified, it should default to 1.

maxSlots

attribute long maxSlots

Maximum number of requested slots. If this attribute is not specified, it should default to the value of minSlots attribute or to 1 if the minSlots attribute is not specified.

candidateMachines

attribute StringList candidateMachines

Requests that the reservation must be created on the any subset of the given machines. If this attribute is not specified, it should default to the result of `MonitoringSession::drmMachineNames`.

nativeOptions

attribute string nativeOptions

This string attribute allows the DRMAA library user to pass DRMS-specific native options for the advance reservation. One example is the definition of a proper job queue name for the reservation. As far as the DRMAA interface specification is concerned, the `nativeOptions` string attribute is an implementation-defined string and is interpreted by each DRMAA library in its specific way. Care SHOULD be exercised to not change the reservation parameters semantics, pass options that conflict the already set attributes, or violate the DRMAA API in any way. Implementations are free to reject reservation templates with invalid native specifications.

SessionManager

SessionManager interface

The `SessionManager` interface is the main interface for establishing communications with a given DRM systems. Through the `SessionManager` interface, sessions for job management, monitoring, and/or reservation management can be managed. Additionally, information about the

An application can open more than one DRMAA job session at a time, but every submitted job MUST only belong to one job session. Sessions, in terms of their job lists, are persisted by the library implementation or the DRMS itself (if supported) after closing the session through the `SessionManager.closeJobSession` method. The information is kept until the session is explicitly reaped by the `SessionManager.destroyJobSession` method. If an implementation runs out of resources for storing the session information, the closing function SHOULD throw a `SessionManagementException`.

Job sessions can be re-opened after they were closed. The re-opening of a session MUST be possible on the machine where the job was originally created. Implementations MIGHT also offer to re-open sessions on another machine. If jobs terminate after closing a session, their termination information SHOULD be available when the jobs original session is re-opened, unless the job information has been purged for space constraints or other restrictions. The session name is generated by the DRMAA implementation and can also be queried from the Session interface. Multiple concurrent sessions are allowed.

systemInfo

readonly attribute string systemInfo

A system identifier denotes a specific type of DRM system, e.g. Sun Grid Engine, or other type of system, e.g. DRMAA meta-interface. It allows the application to rely on implementation-specific attributes.

version

readonly attribute Version version

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 `SessionManager` interface. The string of a `Version` instance MUST be of the form "`<major>.<minor>`".

- `major`: This attribute SHALL contain the major version number.
- `minor`: This attribute SHALL contain the minor version number.

createJobSession

JobSession createJobSession(in string contactString, in string sessionName) raises (???)

The `createJobSession` method creates a new `JobSession` object for the application, which allows to submit, monitor and control a group of jobs. The method MUST do whatever work is required to initialize a DRMAA job session for use, for example by connecting the DRMAA library to a DRMS.

The `contactString` parameter is an implementation-dependent string that may be used to specify which DRM system instance to use. 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'. The strings are always implementation dependent and SHOULD NOT be interpreted by the application. Contact strings need to be figured out by the application user manually for every DRMS installation the application is executed upon. 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, the `createJobSession` method SHALL throw a `NoDefaultContactStringSelectedException` or return a corresponding error code if exceptions aren't supported.

The `sessionName` parameter denotes a specific job session name to be used for the new session. If a job session with such a name was created before, the method MUST throw an `InvalidArgumentException`. In all cases, including if the provided name is an empty string or `null`, a new job session MUST be created.

> Thread Safety
> 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 `createJobSession` operation, in order to ensure thread-safe operations for all other job-related DRMAA methods.

openJobSession

JobSession openJobSession(in string sessionName) raises (???)

The `openSession` function is used to open a session that has previously been created, as named by the `sessionName` parameter. The session could have been created by the same application or by a different application running on the same machine or on a different machine. The session could have been previously closed or may still be active. If no session with the given `sessionName` exists, an `InvalidArgumentException` will be raised.

closeJobSession

void closeJobSession(in JobSession s) raises (???)

The `closeJobSession` method MUST do whatever work is required to disengage from the DRM system and finally persist the list of jobs in the session to some stable storage. This method SHALL NOT affect any jobs in the session (e.g., queued and running jobs remain queued and running). Any job template instances which have not yet been deleted become invalid after `closeJobSession` is called, even after a subsequent call to `createJobSession`.

> Thread Safety
> The `closeJobSession` method SHOULD be called only once, by only one of the application threads. This needs to be ensured by the library implementation. Additional calls to `closeJobSession` beyond the first SHOULD throw a `NoActiveSessionException` or return a corresponding error code if exceptions aren't supported.

destroyJobSession

void destroyJobSession(in string sessionName) raises (???)

> Make it very clear that only this method removes (meaning reaps) job information, but does not kill running jobs in the DRM. Maybe this should be also stated in the description of the Job interface.

> Thread Safety

getJobSessions

StringList getJobSessions() raises (???)

> Thread Safety

createMonitoringSession

MonitoringSession createMonitoringSession (in string contactString) raises (???)

> Thread Safety

closeMonitoringSession

void closeMonitoringSession(in MonitoringSession s) raises (???)

createReservationSession

ReservationSession createReservationSession(in string sessionName, in string contactString) raises (???)

openReservationSession

ReservationSession openReservationSession(in string sessionName) raises (???)

closeReservationSession

void closeReservationSession(in ReservationSession s) raises (???)

destroyReservationSession

void destroyReservationSession(in string sessionName) raises (???)

getReservationSessions

StringList getReservationSessions() raises (???)

Thread Safety

DRMAA2 JSDL Profile

Security Considerations

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.

References

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.

Acknowledgements

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, Mariusz Mamonski, 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.

Suggested Configuration Names

This pages hosts the DRMAA proposals for standardized configuration names. It is unlikely that this list ends up in the real specification - instead, we intend to maintain it on drmaa.org as "working group recommendation". Implementations shall be never mandated to support any of this.

All attributes are expected to work on the execution host.

The April 14th DRMAA telco concluded that the group should only specify the configuration name suggestions, since installation aspects are truly site-specific.

Configuration Name	Source	Description	Expected Environment Variables	Expected Binaries in Path	Expected Libraries in LD_LIBRARY_PATH	Expected Files
MPI	GFD.115	Any MPI environment	??	mpirun	??	??
GridMPI	GFD.115	GridMPI environment	??	??	??	??
IntelMPI	GFD.115	Intel MPI environment	??	??	??	??
LAM-MPI	GFD.115	LAM/MPI environment	??	??	??	??
MPICH1	GFD.115	MPICH Version 1 environment	??	??	??	??
MPICH2	GFD.115	MPICH Version 2 environment	??	??	??	??
MPICH-GM	GFD.115	MPICH-GM environment	??	??	??	??
MPICH-MX	GFD.115	MPICH-MX environment	??	??	??	??
MVAPICH	GFD.115	MVAPICH (MPI-1) environment	??	??	??	??
MVAPICH2	GFD.115	MVAPICH2 (MPI-2) environment	??	??	??	??
OpenMPI	GFD.115	Open MPI environment	??	??	??	??
POE	GFD.115	POE (IBM MPI) environment	??	??	??	??
PVM	GFD.115	Parallel Virtual Machine environment	??	??	??	??
OpenMP	DRMAA WG	OpenMP ennvironment	OMP_NUM_THREADS	??	??	??
Java	DRMAA WG	Java environment	JAVA_HOME	java, javacc	??	??