Peter Tröger, University of Potsdam (maintainer)

Distributed Resource Management Application API Bindings for .NET v0.2

Status of This Memo

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

Copyright Notice

Abstract

This document describes the language binding of the Distributed Resource Management Application API for the .NET framework. The document is based on the DRMAA 1.0 specification [DRMAA], available as GFD.022 recommendation document from GGF.

Content

2. Introduction

This document describes the .NET language binding for the DRMAA (GFD.022) specification. It is described with the C# programming language, which is specified in the ECMA-334 standard [ECMA 334]. It relies on .NET features as defined in the Common Language Infrastructure (CLI) specification in the ECMA-335 standard document [ECMA 335].

The .NET binding was developed with the Microsoft .NET 1.1 framework, but can be implemented with all ECMA-compliant .NET implementations.

This specification conforms to the Common Language Specification (CLS) rules as defined in ECMA-335, which ensures the .NET programming language interoperability.

2.1 How to read this document

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

This document covers only the functional mapping of the DRMAA specification to a .NET environment. The semantics of functions, parameters and error conditions are specified in the DRMAA standard document. The text refers to the respective chapter in the DRMAA standard whenever it is necessary.

2.2 Design Decisions

The DRMAA specification was designed with a C language implementation in mind. The mapping to an object oriented environment leads to several changes in naming, error handling and interface design.

The method and parameter names of the DRMAA specification are converted to the “Pascal case” naming convention, according to the Microsoft .NET design guidelines for class library developers. The prefixed “drmaa_” string is removed from the function names - the namespace concept ensures a distinction of .NET DRMAA function names to other libraries.
The naming of job template parameters was also changed with respect to the .NET naming conventions. Parameter names are replaced where they conflict with reserved .NET keywords.

The DRMAA function set for job status queries is mapped to the *JobTerminationInfo* class.

Error codes are represented by exceptions, therefore all functions return either one of there OUT values or nothing. The *drmaa_context_error_buf* parameter is not necessary in the .NET mapping, since every exception object has a *Text()* method that returns the textual representation of the error.

3. Library Assembly

A .NET DRMAA implementation MUST assign the CLS-compliance attribute to the resulting library assembly:

```
[assembly:CLSCompliant(true)]
```

It declares the overall assembly as compliant to the Common Language Specification, which enables the proper usage of the library with all compliant .NET languages. The .NET DRMAA implementation MUST be build with a compiler that treats CLS violation as a compilation error.

4. Namespace

A .NET implementation of DRMAA MUST be encapsulated in the *Drmaa* namespace:

```
namespace Drmaa
{
    // Exceptions
    // Enumerations
    // Classes
    // IDrmaa interface
}
```

5. Data structures

5.1 Exceptions

The error codes from the DRMAA specification[1] are directly mapped to exception classes in .NET, according to the naming conventions described in 2.2. All exception classes inherit from the *DrmaaException* class, which inherits from the .NET *ApplicationException* class:

```
public abstract class DrmaaException : ApplicationException

public class InternalErrorException : DrmaaException
public class DrmCommunicationFailureException : DrmaaException
public class AuthFailureException : DrmaaException
public class InvalidArgumentException : DrmaaException
public class InvalidContactStringException : DrmaaException
public class DefaultContactStringErrorException : DrmaaException
public class NoDefaultContactStringSelectedException : DrmaaException
public class DrmsInitFailedException : DrmaaException
public class AlreadyActiveSessionException : DrmaaException
public class NoActiveSessionException : DrmaaException
public class DrmsExitErrorException : DrmaaException
public class InvalidAttributeFormatException : DrmaaException
public class InvalidAttributeValueException : DrmaaException
public class ConflictingAttributeValuesException : DrmaaException
public class TryLaterException : DrmaaException
public class DeniedByDrmException : DrmaaException
public class InvalidJobException : DrmaaException
public class ResumeInconsistentStateException : DrmaaException
public class SuspendInconsistentStateException : DrmaaException
public class HoldInconsistentStateException : DrmaaException
public class ReleaseInconsistentStateException : DrmaaException
public class ExitTimeoutException : DrmaaException
public class NoRUsageException : DrmaaException
```

The mapping of DRMAA job templates to the *JobTemplate* class leads to the need for an additional exception. It occurs when a *JobTemplate* object is invalidated with the *IDrmaa.DeleteJobTemplate()* method and accessed afterwards:

```
public class InvalidJobTemplateException : DrmaaException
```

The DRMAA error code *DRMAA_ERRNO_NO_MEMORY* is represented by the .NET *OutOfMemoryException* class.

[1] See DRMAA chapter 3.3

Every exception MUST override the *Message()* method and return an error text with not more than 1024 characters length[2]. Additional parameterized constructors are not needed. The library MAY support other public members in the exception class implementation.

All exception classes MUST be marked with the *[Serializable]* attribute.

5.2 Enumerations

The DRMAA specification defines constant values in different scopes. They are mapped to .NET enumerations. The reader should refer to the DRMAA specification for their semantic meaning.

File transfer mode in a job template

```
public enum TransferFileMode          // drmaa_transfer_files[3]
{
      NoTransfer,                     // ""
      TransferInput,                  // "i"
      TransferOutput,                 // "o"
      TransferError,                  // "e"
      TransferInputAndOutput,         // "io"
      TransferInputAndError,          // "ie"
      TransferOutputAndError,         // "oe"
      TransferInputAndOutputAndError  // "ioe"
};
```

Job state at submission time

```
public enum JobState            // drmaa_js_state[4]
{
      Active,                   // "drmaa_active"
      Hold                      // "drmaa_hold"
};
```

Status code for submitted jobs

```
public enum JobProgramStatus    // 'OUT remote_ps' in
                                      drmaa_job_ps() method[5]
{
      Undetermined = 0x00,      // DRMAA_PS_UNDETERMINED
      QueuedActive = 0x10,      // DRMAA_PS_QUEUED_ACTIVE
      SystemOnHold = 0x11,      // DRMAA_PS_SYSTEM_ON_HOLD
      UserOnHold = 0x12,        // DRMAA_PS_USER_ON_HOLD
      UserSystemOnHold = 0x13,  // DRMAA_PS_USER_SYSTEM_ON_HOLD
      Running = 0x20,           // DRMAA_PS_RUNNING
      SystemSuspended = 0x21,   // DRMAA_PS_SYSTEM_SUSPENDED
      UserSuspended = 0x22,     // DRMAA_PS_USER_SUSPENDED
      Done = 0x30,              // DRMAA_PS_DONE
      Failed = 0x40             // DRMAA_PS_FAILED
};
```

Control action for submitted jobs

[2] See DRMAA chapter 3.2
[3] See DRMAA chapter 3.2.4
[4] See DRMAA chapter 3.2.3
[5] See DRMAA chapter 3.2.6

```
public enum JobControlAction    // 'IN action' in
                                      drmaa_control() method[6]
{
      Suspend,                  // DRMAA_CONTROL_SUSPEND
      Resume,                   // DRMAA_CONTROL_RESUME
      Hold,                     // DRMAA_CONTROL_HOLD
      Release,                  // DRMAA_CONTROL_RELEASE
      Terminate                 // DRMAA_CONTROL_TERMINATED
};
```

Termination reason for a job6

```
public enum JobTerminationReason
{
      Exited,         // drmaa_wifexited()    : exited != 0
      Signaled,       // drmaa_wifsignaled() : signaled != 0
      Aborted         // drmaa_wifaborted()   : aborted != 0
};
```

6. Job Termination Information

The DRMAA specification defines the acquisition of job termination information as a two-stage process - the *drmaa_wait* function returns a general status code, which must be further evaluated by a set of functions6. The .NET DRMAA mapping realizes this as one-step procedure, where the returned status object consists of all possibly available information. The status is defined with the *JobTerminationInfo* class, which represents the result type of an *IDrmaa.Wait()* call:

```
public class JobTerminationInfo
{
      public JobTerminationReason Reason {get;}  // see above
      public int ExitStatus {get;}        // drmaa_wexitstatus()
      public string TermSig {get;}        // drmaa_wtermsig()
      public bool HasCoreDump {get;}      // drmaa_wcoredump()
}
```

According to the DRMAA specification *TermSig* returns the string representation of the signal that caused the job termination.
If *JobTerminationInfo.Reason != Exited* and an application fetches the value of *ExitStatus*, the implementation SHOULD throw an *InvalidArgumentException*. If *JobTerminationInfo.Reason != Signaled* and an application fetch the value of *TermSig*, the implementation SHOULD also throw an *InvalidArgumentException*. [7]

7. Job Identifier

An implementation MUST define a *JobIdentifier* class, which MUST contain an overloaded version of the *Object.ToString()* method:

```
public class JobIdentifier
{
      public override string ToString() {…};
}
```

It is possible for the implementer to add library-specific functionality to the class, however it has to be ensured that *ToString()* always returns an unique string for the particular *JobIdentifier* object.[8]

[6] See DRMAA chapter 3.2.6
[7] Please note that the error code for this situation is not defined in the DRMAA specification.
[8] See DRMAA chapter 3.2.5

8. Timestamps

The DRMAA specification supports the notion of partial timestamps for job template attributes, including optional time zone information. Since .NET only supports *DateTime* objects without time zone information, it is necessary to offer conversion mechanisms for interoperability with other .NET code. This MUST be realized with a special *Timestamp* class that has the following layout:

```
public class Timestamp: IComparable
{
      public string DrmaaString {get;}
      public DateTime LocalDateTime {get;}
      public DateTime UtcDateTime {get;}
      public int CompareTo(object obj);
      public static Timestamp FromDrmaaString(string drmaaTime);
      public static Timestamp FromLocalDateTime(DateTime dt);
      public static Timestamp FromUtcDateTime(DateTime dt);
      public int CompareTo(object obj);
}
```

The *FromDrmaaString(), FromLocalDateTime()* and *FromUtcTime()* functions create a valid *Timestamp* object for the given parameter, which is either a .NET *DateTime* object or a DRMAA compliant string.

The handling of incomplete time specification MUST be realized according to the DRMAA 1.0 specification[9]. The completion of the values MUST be done every time one of the properties *LocalDateTime* or *UtcDateTime* is read.

The usage of an invalid DRMAA string with the *FromDrmaaString* method MUST lead to an *InvalidAttributeFormatException* from the method.

The *DrmaaString* property SHOULD return the original DRMAA time string the object was created with.

An implementation SHOULD support the *IComparable* interface for *Timestamp* objects. In that case it SHOULD also override *System.Object.Equals* and the operators *"==", "<", ">", "<=", ">="* to ensure consistency to it's *IComparable* interface implementation.

9. Job Templates

The .NET DRMAA binding defines a DRMAA job template as class with the following layout:

```
public class JobTemplate
{
      public const string hostHomeDir = "$drmaa_hd_ph$";
      public const string jobIndex = "$drmaa_incr_ph$";
      public const string workingDir = "$drmaa_wd_ph$";
      public string RemoteCommand {get; set;}
      public string[] InputParameters {get; set;}
      public JobState JobSubmissionState {get; set;}
      public IDictionary JobEnvironment {get; set;}
      public string WorkingDirectory  {get; set;}
      public string JobCategory  {get; set;}
      public string NativeSpecification  {get; set;}
      public string[] Email   {get; set;}
      public bool BlockEmail   {get; set;}
      public Timestamp StartTime  {get; set;}
      public string JobName   {get; set;}
```

[9] See DRMAA chapter 3.2.3

```
        public string InputPath  {get; set;}
        public string OutputPath   {get; set;}
        public string ErrorPath   {get; set;}
        public bool JoinFiles  {get; set;}
        [JobTemplateOptional]
        public TransferFileMode TransferFiles {get; set;}
        [JobTemplateOptional]
        public Timestamp DeadlineTime {get; set;}
        [JobTemplateOptional]
        public TimeSpan HardWallclockTimeLimit {get; set;}
        [JobTemplateOptional]
        public TimeSpan SoftWallclockTimeLimit {get; set;}
        [JobTemplateOptional]
        public TimeSpan HardRunDurationLimit {get; set;}
        [JobTemplateOptional]
        public TimeSpan SoftRunDurationLimit {get; set;}
    };
```

The three constants *hostHomeDir*, *jobIndex* and *workingDir* define dynamic placeholder strings as defined in the DRMAA specification[10].

An implementation MAY define a *JobTemplateOptional* attribute with the following layout:

```
[AttributeUsage( AttributeTargets.Field|AttributeTargets.Property,
                 Inherited=true)]
class JobTemplateOptionalAttribute: Attribute
{
}
```

If defined, the attribute MUST be assigned to all the members of the *JobTemplate* class implementation that represent optional job template attributes[11]. This allows the introspective analysis of a job template class for the nature of its parameters. Please note that the explicit determination of available optional attributes is not covered by the DRMAA specification, therefore this attribute is optional.

If an implementation does not support an optional parameter, then the usage of the regarding setter / getter methods MUST always lead to an *InvalidArgumentException*.

The *SetAttribute* / *GetAttribute* methods from the original specification are also supported in the library interface in order to support the usage of DRMAA string representations for attribute values. The correspondence of DRMAA attribute names with the class members is shown in the following table:

DRMAA attribute name	.NET JobTemplate property	Type
drmaa_remote_command	RemoteCommand	System.String
drmaa_js_state	JobSubmissionState	Drmaa.JobState
drmaa_wd	WorkingDirectory	System.String
drmaa_job_category	JobCategory	System.String
drmaa_native_specification	NativeSpecification	System.String
drmaa_block_email	BlockEmail	System.Boolean
drmaa_start_time	StartTime	Drmaa.Timestamp
drmaa_job_name	JobName	System.String
drmaa_input_path	InputPath	System.String
drmaa_output_path	OutputPath	System.String
drmaa_error_path	ErrorPath	System.String
drmaa_join_files	JoinFiles	System.Boolean

[10] See DRMAA chapter 3.2.3
[11] See DRMAA chapter 3.2.4

drmaa_transfer_files	TransferFiles	Drmaa.TransferFileMode
drmaa_deadline_time	DeadlineTime	Drmaa.Timestamp
drmaa_wct_hlimit	HardWallclockTimeLimit	System.TimeSpan
drmaa_wct_slimit	SoftWallclockTimeLimit	System.TimeSpan
drmaa_duration_hlimit	HardRunDurationLimit	System.TimeSpan
drmaa_duration_slimit	SoftRunDurationLimit	System.TimeSpan
drmaa_v_argv	InputParameters	System.String[]
drmaa_v_env	JobEnvironment	System.Collections.IDictionary
drmaa_v_email	Email	System.String[]

10. Mapping for functions - The IDrmaa interface

The *IDrmaa* interface contains all functional methods from the DRMAA interface specification.

An implementation SHOULD provide the interface in a class with the name *Drmaa*, allowing an easy exchange of the .NET DRMAA library without changing the application code. This is the usual solution for “pluggable” libraries in .NET in contrast the Java with it’s factory approach.

The interface layout is defined as follows; possible exceptions for the methods are described as inline code documentation:

```
public interface IDrmaa
{
    JobIdentifier SessionAll {get;}
    JobIdentifier SessionAny {get;}

    /// <exception cref="Drmaa.AlreadyActiveSessionException"/>
    /// <exception cref="Drmaa.InvalidContactStringException"/>
    /// <exception cref="Drmaa.DefaultContactStringErrorException"/>
    /// <exception cref="System.OutOfMemoryException"/>
    /// <exception cref="Drmaa.InternalErrorException"/>
    void Init(string contact);

    /// <exception cref="Drmaa.DrmsExitErrorException"></exception>
    /// <exception cref="Drmaa.NoActiveSessionException"></exception>
    /// <exception cref="Drmaa.InternalErrorException"/>
    void Exit();

    /// <exception cref="Drmaa.DrmCommunicationFailureException"/>
    /// <exception cref="Drmaa.InternalErrorException"/>
    /// <exception cref="System.OutOfMemoryException"/>
    /// <exception cref="Drmaa.InternalErrorException"/>
    JobTemplate AllocateJobTemplate();

    /// <exception cref="Drmaa.DrmCommunicationFailureException"/>
    /// <exception cref="Drmaa.InternalErrorException"/>
    void DeleteJobTemplate(JobTemplate jt);

    /// <exception cref="Drmaa.InvalidAttributeFormatException"/>
    /// <exception cref="Drmaa.InvalidAttributeValueException"/>
    /// <exception cref="Drmaa.ConflictingAttributeValuesException"/>"
    /// <exception cref="Drmaa.InvalidJobTemplateException"/>"
    /// <exception cref="Drmaa.InvalidArgumentException"/>
    /// <exception cref="System.OutOfMemoryException"/>
    /// <exception cref="Drmaa.InternalErrorException"/>
    void SetAttribute(JobTemplate jt, string name, string value);

    /// <exception cref="Drmaa.InvalidAttributeValueException"/>
    /// <exception cref="Drmaa.InvalidJobTemplateException"/>"
```

```
    /// <exception cref="Drmaa.InternalErrorException"/>
    string GetAttribute(JobTemplate jt, string name);

    /// <exception cref="Drmaa.InvalidAttributeFormatException"/>
    /// <exception cref="Drmaa.InvalidAttributeValueException"/>
    /// <exception cref="Drmaa.ConflictingAttributeValuesException"/>
    /// <exception cref="Drmaa.InvalidJobTemplateException"/>"
    /// <exception cref="System.OutOfMemoryException"/>
    /// <exception cref="Drmaa.InternalErrorException"/>
    void SetVectorAttribute(JobTemplate jt, string name,
                            string[] values);

    /// <exception cref="Drmaa.InvalidAttributeValueException"/>
    /// <exception cref="Drmaa.InvalidJobTemplateException"/>"
    /// <exception cref="Drmaa.InternalErrorException"/>
    string[] GetVectorAttribute(JobTemplate jt, string name);

    /// <exception cref="System.OutOfMemoryException"/>
    /// <exception cref="Drmaa.InternalErrorException"/>
    string[] GetAttributeNames();

    /// <exception cref="System.OutOfMemoryException"/>
    /// <exception cref="Drmaa.InternalErrorException"/>
    string[] GetVectorAttributeNames();

    /// <exception cref="Drmaa.TryLaterException"/>
    /// <exception cref="Drmaa.DeniedByDrmException"/>
    /// <exception cref="Drmaa.DrmCommunicationFailureException"/>
    /// <exception cref="Drmaa.AuthFailureException"/>
    /// <exception cref="Drmaa.InternalErrorException"/>
    JobIdentifier RunJob(JobTemplate jt);

    /// <exception cref="Drmaa.TryLaterException"/>
    /// <exception cref="Drmaa.DeniedByDrmException"/>
    /// <exception cref="Drmaa.DrmCommunicationFailureException"/>
    /// <exception cref="Drmaa.AuthFailureException"/>
    /// <exception cref="Drmaa.InternalErrorException"/>
    JobIdentifier[] RunBulkJobs(JobTemplate jt, int startIndex,
                                int endIndex, int incIndex);

    /// <exception cref="Drmaa.DrmCommunicationFailureException"/>
    /// <exception cref="Drmaa.AuthFailureException"/>
    /// <exception cref="Drmaa.ResumeInconsistentStateException"/>
    /// <exception cref="Drmaa.SuspendInconsistentStateException"/>
    /// <exception cref="Drmaa.HoldInconsistentStateException"/>
    /// <exception cref="Drmaa.ReleaseInconsistentStateException"/>
    /// <exception cref="Drmaa.InvalidJobException"/>
    /// <exception cref="Drmaa.InternalErrorException"/>
    void Control (JobIdentifier jobId, JobControlAction action);

    /// <exception cref="Drmaa.DrmCommunicationFailureException"/>
    /// <exception cref="Drmaa.AuthFailureException"/>
    /// <exception cref="Drmaa.InvalidJobException"/>
    /// <exception cref="Drmaa.InternalErrorException"/>
    bool Synchronize (JobIdentifier[] jobIds, TimeSpan timeout,
                      bool dispose);
```

```
    /// <exception cref="Drmaa.DrmCommunicationFailureException"/>
    /// <exception cref="Drmaa.AuthFailureException"/>
    /// <exception cref="Drmaa.NoRUsageException"/>
    /// <exception cref="Drmaa.ExitTimeoutException"/>
    /// <exception cref="Drmaa.InvalidJobException"/>
    /// <exception cref="Drmaa.InternalErrorException"/>
    JobIdentifier Wait (JobIdentifier jobId, TimeSpan timeout,
                        out JobTerminationInfo stat,
                        out IDictionary rusage);

    /// <exception cref="Drmaa.DrmCommunicationFailureException"/>
    /// <exception cref="Drmaa.AuthFailureException"/>
    /// <exception cref="Drmaa.InvalidJobException"/>
    /// <exception cref="Drmaa.InternalErrorException"/>
    JobProgramStatus JobPS(JobIdentifier jobId);

    /// <exception cref="Drmaa.InternalErrorException"/>
    string[] GetContact();

    /// <exception cref="Drmaa.InternalErrorException"/>
    string[] GetDRMSystem();

    /// <exception cref="Drmaa.InternalErrorException"/>
    string[] GetDRMAAImplementation();

    /// <exception cref="Drmaa.InternalErrorException"/>
    void GetVersion(out int major, out int minor);
}
```

The *SessionAll* and *SessionAny* properties of the interface return a special *JobIdentifier* object that can be used with the *Wait()* or *Synchronize()* function[12]. If such an object is modified, the regarding setter method SHOULD throw an *InvalidJobTemplateException*.

The usage of *DeleteJobTemplate*() only causes only an invalidation of the referred *JobTemplate* object. An implementation MUST ensure that a latter use of this object leads to an *InvalidJobTemplateException*.

The *timeout* parameter of *Synchronize()* and *Wait()* can take *TimeSpan.MaxValue* as representation for an indefinite wait and *TimeSpan.Zero* for an immediate return[13].

The parameter names for the *RunBulkJobs* method were changed because of the fact that “end” is a reserved keyword in Visual Basic .NET.

11. .NET Language Binding Example

```
using System;
using System.Threading;
using System.Collections;
using Drmaa;

namespace example {
   class Example {
      static IDrmaa session = null;
```

[12] This maps to DRMAA_JOB_IDS_SESSION_ALL and DRMAA_JOB_IDS_SESSION_ANY from DRMAA chapter 3.2.6.
[13] This maps to DRMAA_TIMEOUT_WAIT_FOREVER and DRMAA_TIMEOUT_NO_WAIT from DRMAA chapter 3.2.6.

```
static JobTemplate CreateTpl(string path, int seconds, bool isBulk) {
   JobTemplate jt = session.AllocateJobTemplate();
   jt.RemoteCommand = path;
   jt.WorkingDirectory = JobTemplate.hostHomeDir;
   jt.InputParameters = new String[] {seconds.ToString()};
   jt.JoinFiles = true;
   if (isBulk)
      jt.OutputPath = JobTemplate.hostHomeDir + "/DRMAA_JOB";
   else
      jt.OutputPath = JobTemplate.hostHomeDir + "/DRMAA_JOB" +
                              JobTemplate.jobIndex;
   return jt;
}

static void Main(string[] args)
{
   ArrayList allIds = new ArrayList();
   IEnumerator e = null;
   bool again;
   session = (IDrmaa)new LocalDrmaa();
   session.Init(null);
   // submit bulk job
   JobTemplate jt = CreateTpl(args[0], 5, true);
   JobIdentifier[] ids = null;
   again = false;
   while (again) {
      try {
         ids = session.RunBulkJobs(jt, 1, 8, 1);
      }
      catch (TryLaterException) {
         again = true;
         Thread.Sleep(1000);
      }
   }
   allIds.AddRange(ids);
   Console.WriteLine("Submitted bulk job with job id's:");
   e = allIds.GetEnumerator();
   while (e.MoveNext()) {
      Console.WriteLine(e.Current.ToString());
   }
   session.DeleteJobTemplate(jt);

   // submit sequential job
   jt = CreateTpl(args[0], 5, false);
   JobIdentifier id = null;
   again = false;
   while (again) {
      try {
         id = session.RunJob(jt);
      }
      catch (TryLaterException) {
         again = true;
         Thread.Sleep(1000);
      }
   }
   Console.WriteLine(id.ToString());
   allIds.Add(id);

   // synchronize with all jobs
   session.Synchronize((JobIdentifier[])allIds.ToArray(typeof(string)),
                              TimeSpan.MaxValue, false);
   Console.WriteLine("Synchronized with all jobs");
```

```
        // wait for all jobs
        e = allIds.GetEnumerator();
        JobTerminationInfo jInfo;
        IDictionary rUsage;
        while (e.MoveNext()) {
           JobIdentifier lastId = session.Wait((JobIdentifier)e.Current,
                                                TimeSpan.MaxValue,
                                                out jInfo, out rUsage);
           // report about termination status
           switch (jInfo.Reason) {
              case JobTerminationReason.Exited:
                    Console.WriteLine("{0} exited with status {1}",
                                      lastId.ToString(), jInfo.ExitStatus);
                    break;
              case JobTerminationReason.Signaled:
                    Console.WriteLine("{0} finished due to signal {1}",
                                      lastId.ToString(), jInfo.TermSig);
                    break;
              case JobTerminationReason.Aborted:
                    Console.WriteLine("{0} was aborted", lastId.ToString());
                    break;
}}}}}
```

12. Security Considerations

A .NET DRMAA implementation SHOULD have a ‘strong name’ in order to be uniquely identifiable by the code access security mechanism. This avoids exchangeability of DRMAA libraries for a compiled application, therefore strong naming is recommended, but not required.

A publisher MAY sign an assembly (“Authenticode”) if he has an appropriate key infrastructure.

An implementation SHOULD define its minimal required permission set as a set of assembly attributes, e.g. to assert security permissions or to read a specific file:

```
[assembly:SecurityPermissionAttribute(SecurityAction.RequestMinimum,
                                      Assertion=true)]

[assembly:FileIOPermission(SecurityAction.RequestMinimum,
                           Read=@”c:\condor\condor.dll”)]
```

A valid implementation MAY consider the permission demands of used class library functions by specifying them also as minimal required assembly security permission. It MAY also refuse unnecessary permissions with the *RequestRefuse* security action.

An implementation MAY assert permissions that are not available for the callers, allowing less-trusted applications to utilize the DRMAA functionality. In this case it SHOULD define and demand a custom permission with the name *DrmaaPermission* for the caller.

Author Information

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

Intellectual Property Statement

Full Copyright Notice

References

[DRMAA] Hrabri Rajic et. al. *Distributed Resource Management Application API Specification 1.0.* http://forge.ggf.org/projects/drmaa-wg/, June 2004.

[ECMA 334] ECMA. *C# Language Specification*, Second Edition, December 2002

[ECMA 335] ECMA. *Common Language Infrastructure (CLI)*, Second Edition, December 2002

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