Ceriel Jacobs, VU

Thilo Kielmann, VU[1]

other contributors??

February 19, 2008

A Simple API for Grid Applications (SAGA)
Java Language Binding

Abstract

This document specifies the language binding to the Java language, hence the Java syntax, for the Simple API for Grid Applications (SAGA Core API), as described in GFD.90. First, design principles are outlined, and Java-specific issues are discussed. Then, the SAGA Core API is rendered in the Java language, following the package structure of the GFD.90 document.

---

[1]editor

Contents

1   Introduction

This document specifies the language binding to the Java language, hence the Java syntax, for the Simple API for Grid Applications (SAGA Core API), as described in GFD.90. First, design principles are outlined, and Java-specific issues are discussed. Then, the SAGA Core API is rendered in the Java language, following the package structure of the GFD.90 document.

The *Simple API for Grid Applications* has been designed as a high-level API that directly addresses the needs of application developers. Its purpose is two-fold:

1. Provide a **simple** API that can be used with much less effort compared to the vanilla interfaces of existing grid middleware. A guiding principle for achieving this simplicity is the *80–20 rule*: serve 80 % of the use cases with 20 % of the effort needed for serving 100 % of all possible requirements.

2. Provide a standardized, common interface across various grid middleware systems and their versions.

Analogously, any language binding of the SAGA API (here, to the Java language) is having two purposes:

1. Provide the syntax of the SAGA API in the target programming language (here: Java).

2. Maintain the typical "look-and-feel" of programming in the target language, thus supporting the recognized best programming practices for the target programming language. According to GFD.90 (Section 2.3), a laguage binding may overrule certain details of the language-neutral SAGA specification (GFD.90) if this supports the acknowledged "best practices" of the programming language.

   For the Java language, we will summarize such deviations from GFD.90 in Section 2.

1.1   How to read this Document

This document is an API *specification*, and as such targets *implementors of the API*, rather than its end users. In particular, this document should not be confused with a SAGA Users' Guide. This document might be useful as an API reference, but, in general, the API users' guide and reference should be published as separate documents, and should accompany SAGA implementations. The

latest version of the users guide and reference can be found at `http://saga.cct.lsu.edu`.

An implementor of the SAGA API should read the complete document carefully. General design considerations of the SAGA API are explained in the GFD.90 document. While this language-binding document prescribes the concrete SAGA syntax in the Java language, GFD.90 needs to be referred to as well. Any implementation in the Java language will be considered SAGA compliant if and only if it follows the syntax (classes, interfaces, etc.) prescribed in this document. In addition, GFD.90, especially Section 2.3, applies.

This document is structured as follows. This Section focusses on the formal aspects of an OGF recommendation document. Section 2 describes the general design considerations, related to the Java language. The subsequent sections are mirroring the packages structure of the SAGA API from GFD.90, prescribing the Java language binding on a per-package basis. Finally, Section 5 gives author contact information and provides disclaimers concerning intellectual property rights and copyright issues, according to OGF policies.

1.2   Notational Conventions

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 [2].

1.3   Security Considerations

As the SAGA API is to be implemented on different types of grid (and non-grid) middleware, it does not specify a single security model, but rather provides hooks to interface to various security models – see the documentation of the `saga::context` class in the GFD.90 document as well as in Section 3.6 for details.

A SAGA implementation is considered secure if and only if it fully supports (i.e. implements) the security models of the middleware layers it builds upon, and neither provides any (intentional or unintentional) means to by-pass these security models, nor weakens these security models' policies in any way.

2   Java-specific Design Considerations

Figure 1 shows all classes and interfaces from the language-independent SAGA specification (GFD.90). In this section, we present the general, Java-specific design considerations followed throughout this document.

2.1   Java language version

The Java language version used is the one provided in J2SE 5.0 [4]. This version of the Java language is widely available and, in contrast to earlier versions, this version provides generics and enumerated types, features used in the SAGA language bindings for Java. Also, this version provides the `java.util.concurrent` package, which is not available in earlier versions, and which is used for the Java language bindings of SAGA tasks.

2.2   Concurrency control and thread-safety

Java SAGA implementations `MUST` be thread-safe. Multiple application threads are allowed to access a common SAGA object. However, no particular order is enforced, unless the application itself does so.

2.3   File I/O and Java file streams

Earlier experience with JavaGAT [10] has shown that having implementations of the Java streams `java.io.InputStream` and `java.io.OutputStream` is very much appreciated by Java application programmers, since these are the types on which most Java I/O is based. Therefore, it was decided to add specifications for `FileInputStream` and `FileOutputStream` to the file package. The `file` class as specified in the SAGA specifications is also specified in the Java language bindings. Of course, implementations and factories may throw the `NotImplemented` exception when methods or complete classes cannot be implemented.

2.4   Files and error handling

The SAGA specifications refer to POSIX error return codes for several methods. However, it is not to be expected that these will be available in existing Java Grid middleware. Also, in Java, error conditions are supposed to be passed on by means of exceptions. Therefore, it was decided that where the SAGA specifications refer to POSIX error codes, a `java.io.IOException` is to be

thrown in these cases. This is less informative than a specific error code, but is more "Java-like", and probably better implementable on existing Java Grid middleware.

2.5 Features unavailable in Java: permissions, links

For some features of the SAGA specifications, most notably permissions and links, Java just does not provide building blocks, even locally. Java is not a systems programming language. Nevertheless, all methods concerning links and permissions are specified in the Java language bindings. SAGA application writers should not be surprised, however, if SAGA implementations throw an `NotImplemented` exception when these methods are invoked.

2.6 Prescribing the API for applications and SAGA implementors

The aim of having bindings of the SAGA API specification to certain programming languages is to define the precise syntax and semantics of the SAGA functionality, in the given language. This language binding can be seen as a contract between applications and SAGA implementors: both parties can safely assume that exactly the classes and interfaces described in this document will be either provided or requested for.

For facilitating both application writing and implementing SAGA, providing the Java language binding in the form of directly usable files is considered important. It has been decided to provide both interfaces and classes from the language-independent SAGA specification in the form of Java *interfaces*. This leaves SAGA implementations with the task of writing classes that implement these interfaces. For allowing applications to create SAGA objects, the interfaces are accompanied by factory classes. Java implementations of SAGA `MUST` implement the interfaces and factories as specified here.

Besides the description in this document, the Java language bindings are provided as Java source files. They are available from **TODO: where???** and can be used for both implementation purposes as well as for generating Javadoc documentation.

This setup requires a bootstrap mechanism for creating factory objects. This is realized with the `SagaFactory` interface, and the `ImplementationBootstrapLoader` class, as described here. This mechanism uses the `saga.factory` system property, to be set by the user to point to an implementation-specific metafactory, which in turn has methods to create factories for all SAGA packages.

2.6.1   SagaFactory

```
package org.ogf.saga.bootstrap;

import org.ogf.saga.buffer.BufferFactory;
import org.ogf.saga.context.ContextFactory;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.file.FileFactory;
import org.ogf.saga.job.JobFactory;
import org.ogf.saga.logicalfile.LogicalFileFactory;
import org.ogf.saga.monitoring.MonitoringFactory;
import org.ogf.saga.namespace.NSFactory;
import org.ogf.saga.rpc.RPCFactory;
import org.ogf.saga.session.SessionFactory;
import org.ogf.saga.stream.StreamFactory;
import org.ogf.saga.task.TaskFactory;


/**
 * This interface must be implemented by a SAGA implementation that uses
 * these language bindings. It creates factories for all packages in SAGA.
 * See the {link ImplementationBootstrapLoader} description.
 */
public interface SagaFactory {

    /**
     * Creates a factory for the Saga buffer package.
     * return the buffer factory.
     * exception NotImplemented is thrown when buffers are not implemented.
     */
    BufferFactory createBufferFactory() throws NotImplemented;

    /**
     * Creates a factory for the Saga context package.
     * return the context factory.
     */
    ContextFactory createContextFactory();

    /**
     * Creates a factory for the Saga file package.
     * Note: this method cannot throw NotImplemented, because the
     * IOVec constructor from the SAGA specs does not throw NotImplemented.
     * return the File factory.
     */
    FileFactory createFileFactory();

    /**
     * Creates a factory for the Saga jobs package.
     * return the jobs factory.
     * exception NotImplemented is thrown when jobs are not implemented.
```

```
 */
JobFactory createJobFactory() throws NotImplemented;

/**
 * Creates a factory for the Saga logical file package.
 * return the logical file factory.
 * exception NotImplemented is thrown when logical file is not implemented.
 */
LogicalFileFactory createLogicalFileFactory() throws NotImplemented;

/**
 * Creates a factory for the Saga monitoring package.
 * return the monitoring factory.
 * exception NotImplemented is thrown when monitoring is not implemented.
 */
MonitoringFactory createMonitoringFactory() throws NotImplemented;

/**
 * Creates a factory for the Saga namespace package.
 * return the namespace factory.
 * exception NotImplemented is thrown when namespaces are not implemented.
 */
NSFactory createNamespaceFactory() throws NotImplemented;

/**
 * Creates a factory for the Saga RPC package.
 * Note: this method cannot throw NotImplemented, because the
 * Parameter constructor from the SAGA specs does not throw NotImplemented.
 * return the RPC factory.
 */
RPCFactory createRPCFactory();

/**
 * Creates a factory for the Saga session package.
 * return the session factory.
 */
SessionFactory createSessionFactory();

/**
 * Creates a factory for the Saga stream package.
 * return the stream factory.
 * exception NotImplemented is thrown when streams are not implemented.
 */
StreamFactory createStreamFactory() throws NotImplemented;

/**
 * Creates a factory for the Saga task package.
 * return the task factory.
 * exception NotImplemented is thrown when tasks are not implemented.
 */
```

```
    TaskFactory createTaskFactory() throws NotImplemented;
}
```

2.6.2   ImplementationBootstrapLoader

```
package org.ogf.saga.bootstrap;

import java.lang.reflect.InvocationTargetException;

import org.ogf.saga.buffer.BufferFactory;
import org.ogf.saga.context.ContextFactory;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.SagaError;
import org.ogf.saga.file.FileFactory;
import org.ogf.saga.job.JobFactory;
import org.ogf.saga.logicalfile.LogicalFileFactory;
import org.ogf.saga.monitoring.MonitoringFactory;
import org.ogf.saga.namespace.NSFactory;
import org.ogf.saga.rpc.RPCFactory;
import org.ogf.saga.session.SessionFactory;
import org.ogf.saga.stream.StreamFactory;
import org.ogf.saga.task.TaskFactory;

/**
 * The idea of this class is that the SAGA user sets the environment variable
 * "saga.factory" to the classname of an implementation of the
 * {link SagaFactory} interface.
 * The ImplementationBootstrapLoader instantiates exactly one instance of
 * this class, which must have a public parameter-less constructor.
 */
public class ImplementationBootstrapLoader {

    private static final String PROPERTY_NAME = "saga.factory";

    private static SagaFactory factory;

    private static synchronized void initFactory() {

        if (factory == null) {
            // Obtain the name of the SAGA factory.
            String factoryName = System.getProperty(PROPERTY_NAME);
            if (factoryName == null) {
                throw new SagaError("No SAGA factory name specified");
            }

            // Try to obtain a class instance of it, using the current
            // class loader, and running the static initializers.
            Class<?> factoryClass;
```

```
            try {
                factoryClass = Class.forName(factoryName);
            } catch(ClassNotFoundException e) {
                throw new SagaError("Could not load class " + factoryName, e);
            }

            // Now try to obtain an instance of this class, using a
            // parameter-less constructor.
            try {
                factory = (SagaFactory) factoryClass.getConstructor()
                    .newInstance();
            } catch(NoSuchMethodException e) {
                throw new SagaError("Factory " + factoryName
                        + " has no public noargs constructor", e);
            } catch(InvocationTargetException e1) {
                throw new SagaError("Constructor of " + factoryName
                        + " threw an exception", e1.getCause());
            } catch(Throwable e2) {
                throw new SagaError("Instantiation of " + factoryName
                        + " failed", e2);
            }
        }
    }

    /**
     * Creates a buffer factory.
     * return a buffer factory.
     * throws NotImplemented is thrown when buffers are not implemented.
     */
    public static BufferFactory createBufferFactory()
        throws NotImplemented {
        initFactory();
        return factory.createBufferFactory();
    }

    /**
     * Creates a context factory. Cannot throw NotImplemented.
     * return a context factory.
     */
    public static ContextFactory createContextFactory() {
        initFactory();
        return factory.createContextFactory();
    }

    /**
     * Creates a file factory. Cannot throw NotImplemented, because the
     * IOVec constructor cannot (according to the SAGA specs).
     * return a file factory.
     */
    public static FileFactory createFileFactory() {
```

```
    initFactory();
    return factory.createFileFactory();
}

/**
 * Creates a job factory.
 * return a job factory.
 * throws NotImplemented is thrown when jobs are not implemented.
 */
public static JobFactory createJobFactory()
    throws NotImplemented {
    initFactory();
    return factory.createJobFactory();
}

/**
 * Creates a logical file factory.
 * return a logical file factory.
 * throws NotImplemented is thrown when logical files are not implemented.
 */
public static LogicalFileFactory createLogicalFileFactory()
    throws NotImplemented {
    initFactory();
    return factory.createLogicalFileFactory();
}

/**
 * Creates a monitoring factory.
 * return a monitoring factory.
 * throws NotImplemented is thrown when monitoring is not implemented.
 */
public static MonitoringFactory createMonitoringFactory()
    throws NotImplemented {
    initFactory();
    return factory.createMonitoringFactory();
}

/**
 * Creates a namespace factory.
 * return a namespace factory.
 * throws NotImplemented is thrown when namespace is not implemented.
 */
public static NSFactory createNamespaceFactory()
    throws NotImplemented {
    initFactory();
    return factory.createNamespaceFactory();
}

/**
 * Creates an RPC factory. Cannot throw NotImplemented,
```

```
 * because the Parameter constructor cannot throw NotImplemented.
 * return an RPC factory.
 */
public static RPCFactory createRPCFactory() {
    initFactory();
    return factory.createRPCFactory();
}

/**
 * Creates a Session factory.
 * return a Session factory.
 */
public static SessionFactory createSessionFactory() {
    initFactory();
    return factory.createSessionFactory();
}

/**
 * Creates a stream factory.
 * return a stream factory.
 * throws NotImplemented is thrown when streams are not implemented.
 */
public static StreamFactory createStreamFactory()
    throws NotImplemented {
    initFactory();
    return factory.createStreamFactory();
}

/**
 * Creates a task factory.
 * return a task factory.
 * throws NotImplemented is thrown when tasks are not implemented.
 */
public static TaskFactory createTaskFactory()
    throws NotImplemented {
    initFactory();
    return factory.createTaskFactory();
}
}
```

2.7   Java language binding overview

The Java language binding consists of the following packages. They are described in detail in the following, referenced sections.

| package	purpose	section
org.ogf.saga	Base Objects	3.2, 3.3
org.ogf.saga.attributes	Attribute Model	3.8
org.ogf.saga.bootstrap	Factory Bootstrapping	2
org.ogf.saga.buffer	I/O Buffer	3.4
org.ogf.saga.context	Context Management	3.6
org.ogf.saga.error	Error Handling	3.1
org.ogf.saga.file	File Management	4.3
org.ogf.saga.job	Job Management	4.1
org.ogf.saga.logicalfile	Replica Management	4.4
org.ogf.saga.monitoring	Monitoring Model	3.9
org.ogf.saga.namespace	Name Spaces	4.2
org.ogf.saga.permissions	Permission Model	3.7
org.ogf.saga.rpc	Remote Procedure Call	4.6
org.ogf.saga.session	Session Management	3.5
org.ogf.saga.stream	Streams	4.5
org.ogf.saga.task	Task Model	3.10
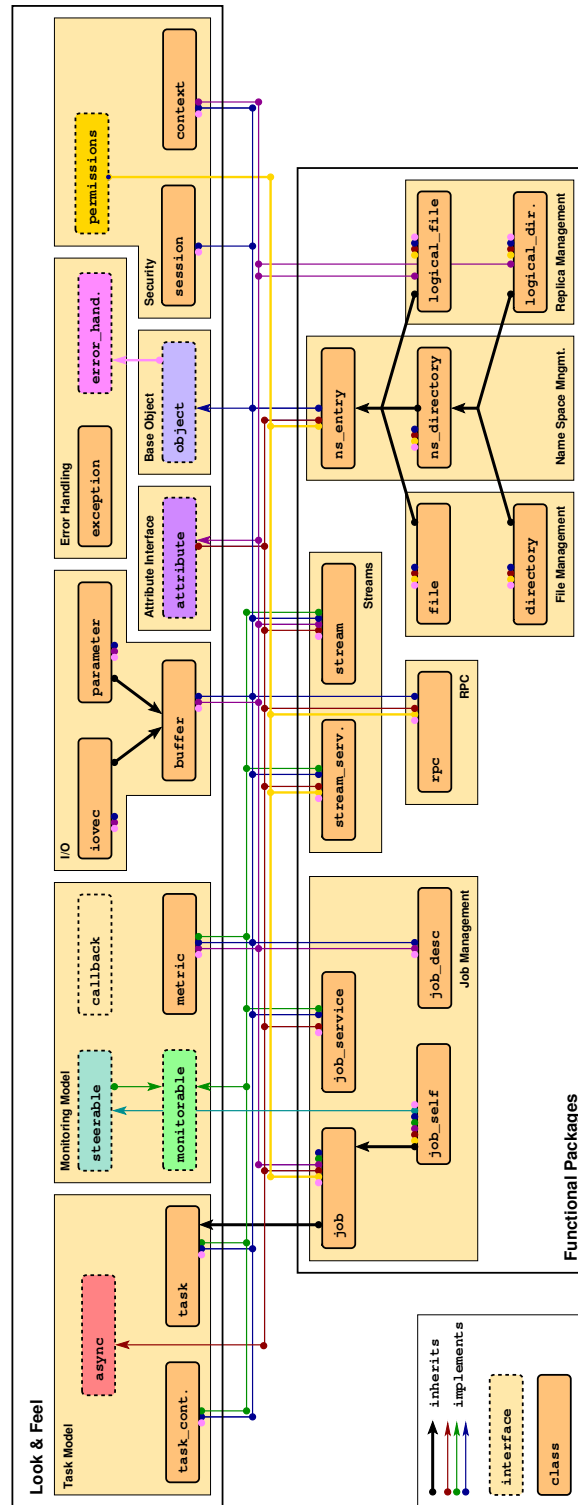
Figure 1: The SAGA class and interface hierarchy, accordign to GFD.90.

3   SAGA API Specification – Look & Feel

The SAGA API consists of a number of interface and class specifications. The relation between these is shown in Figure 1. This figure also marks which interfaces are part of the non-functional, SAGA Look-&-Feel, and which classes are combined into functional packages. The two main parts, non-functional Look-&-Feel and functional packages, are considered to be orthogonally combined with each other. (Future, functional extension packages will implicitly follow the non-functional Look-&-Feel.) This section is presenting the Java rendering of SAGA's non-funtional parts.

3.1   SAGA Error Handling

Each SAGA API call has an associated list of exceptions it may throw. These exceptions all extend the `Exception` class described below.

According to the SAGA specifications, all objects in SAGA implement the `error_handler` interface, which allows a user of the API to query for the latest error associated with a SAGA object (pull). In languages with exception-handling mechanisms, such as Java, C++ and Perl, the language binding MAY allow exceptions to be thrown. If an exception handling mechanism is included in a language binding, the `error handler` MUST NOT be included in the same binding.

Since Java has excellent exception handling support, the Java language bindings use the Java mechanism instead of the `error handler` interface.

For asynchronous operations, any exception occurring during this operation is stored in the task instance performing the operation. This exception can be rethrown using the `rethrow` method of the task instance.

If an error occurs during object creation, then the factory method will throw the corresponding exception.

The SAGA specifications say that in languages bindings where this is appropriate, some API methods return POSIX `errno` codes for errors. This is the case in particular for `read()`, `write()` and `seek()`, for `File` and `Stream`. The respective method descriptions provide explicit details of how `errno` error codes are utilized. In any case, whenever numerical `errno` codes are used, they have to be conforming to POSIX.1 [9].

For Java, this is not appropriate. The "usual" Java `read` and `write` methods are not POSIX-like. Instead, they throw a `java.io.IOException` when an error occurs. This is less informative than a specific error code, but is more "Java-

like", and probably better implementable on existing Java Grid middleware.

3.1.1   Exception

```
package org.ogf.saga.error;

import org.ogf.saga.SagaObject;

/**
 * This is the base class for all exceptions in SAGA.
 * It is a checked exception, so all exceptions in SAGA are
 * checked exceptions.
 */
public abstract class Exception extends java.lang.Exception implements Comparable<Exception> {

    // Determine the order of the exceptions, most specific first.
    protected static final int NOT_IMPLEMENTED = 0;
    protected static final int INCORRECT_URL = 1;
    protected static final int BAD_PARAMETER = 2;
    protected static final int ALREADY_EXISTS = 3;
    protected static final int DOES_NOT_EXIST = 4;
    protected static final int INCORRECT_STATE = 5;
    protected static final int PERMISSION_DENIED = 6;
    protected static final int AUTHORIZATION_FAILED = 7;
    protected static final int AUTHENTICATION_FAILED = 8;
    protected static final int TIMEOUT = 9;
    protected static final int NO_SUCCESS = 10;

    /** Determines how specific the exception is with respect to others. */
    private final int exceptionOrder;

    private static final long serialVersionUID = 1L;

    private transient final SagaObject object;

    /**
     * Constructs a new SAGA exception.
     * param order initializes the exceptionOrder field that determines
     *     which exception is more specific.
     */
    public Exception(int order) {
        object = null;
        exceptionOrder = order;
    }

    /**
     * Constructs a new SAGA exception with the specified detail
     * message.
```

```
 * param order initializes the exceptionOrder field that determines
 *    which exception is more specific.
 * param message the detail message.
 */
public Exception(int order, String message) {
    super(message);
    object = null;
    exceptionOrder = order;
}

/**
 * Constructs a new SAGA exception with the specified cause.
 * param order initializes the exceptionOrder field that determines
 *    which exception is more specific.
 * param cause the cause.
 */
public Exception(int order, Throwable cause) {
    super(cause);
    object = null;
    exceptionOrder = order;
}

/**
 * Constructs a new SAGA exception with the specified detail
 * message and cause.
 * param order initializes the exceptionOrder field that determines
 *    which exception is more specific.
 * param message the detail message.
 * param cause the cause.
 */
public Exception(int order, String message, Throwable cause) {
    super(message, cause);
    object = null;
    exceptionOrder = order;
}

/**
 * Constructs a new SAGA exception with the specified detail
 * message and associated SAGA object.
 * param order initializes the exceptionOrder field that determines
 *    which exception is more specific.
 * param message the detail message.
 * param object the SAGA object associated with the exception.
 */
public Exception(int order, String message, SagaObject object) {
    super(message);
    this.object = object;
    exceptionOrder = order;
}
```

```
    /**
     * Returns the SAGA object associated with this exception.
     * return the associated SAGA object.
     */
    public SagaObject getObject() throws DoesNotExist, NoSuccess {
        if (object == null) {
            throw new DoesNotExist("No object associated with this exception");
        }
        return object;
    }

    /**
     * Returns the message as specified by the SAGA API, i.e.,
     * <exception name>: <message>.
     * return the message.
     */
    public String getMessage() {
        return this.getClass().getSimpleName() + ": "
                + super.getMessage();
    }

    /**
     * Gives preference to the most specific exception.
     * This implements the ordering as in the SAGA specs.
     * Returns < 0 if this exception is more specific than the specified exception,
     * 0 if equal, and > 0 if less.
     */
    public int compareTo(Exception o) {
        return exceptionOrder - o.exceptionOrder;
    }

}
```

3.1.2   NotImplemented

```
package org.ogf.saga.error;

import org.ogf.saga.SagaObject;

/**
 * This exception indicates that a SAGA method is not implemented.
 */
public class NotImplemented extends Exception {

    private static final long serialVersionUID = 1L;

    /**
     * Constructs a NotImplemented exception.
```

```
 */
public NotImplemented() {
    super(NOT_IMPLEMENTED);
}

/**
 * Constructs a NotImplemented exception with the specified detail
 * message.
 * param message the detail message.
 */
public NotImplemented(String message) {
    super(NOT_IMPLEMENTED, message);
}

/**
 * Constructs a NotImplemented exception with the specified cause.
 * param cause the cause.
 */
public NotImplemented(Throwable cause) {
    super(NOT_IMPLEMENTED, cause);
}

/**
 * Constructs a NotImplemented exception with the specified detail
 * message and cause.
 * param message the detail message.
 * param cause the cause.
 *
 */
public NotImplemented(String message, Throwable cause) {
    super(NOT_IMPLEMENTED, message, cause);
}

/**
 * Constructs a NotImplemented exception with the specified detail
 * message and associated SAGA object.
 * param message the detail message.
 * param object the associated SAGA object.
 */
public NotImplemented(String message, SagaObject object) {
    super(NOT_IMPLEMENTED, message, object);
}
}
```

3.1.3 IncorrectURL

```
package org.ogf.saga.error;
```

```
import org.ogf.saga.SagaObject;

/**
 * This exception indicates that a method is given an URL argument that could
 * not be handled.
 */
public class IncorrectURL extends Exception {

    private static final long serialVersionUID = 1L;

    /**
     * Constructs an IncorrectURL exception.
     */
    public IncorrectURL() {
        super(INCORRECT_URL);
    }

    /**
     * Constructs an IncorrectURL exception with the specified detail message.
     * param message the detail message.
     */
    public IncorrectURL(String message) {
        super(INCORRECT_URL, message);
    }

    /**
     * Constructs an IncorrectURL exception with the specified cause.
     * param cause the cause.
     */
    public IncorrectURL(Throwable cause) {
        super(INCORRECT_URL, cause);
    }

    /**
     * Constructs an IncorrectURL exception with the specified detail message
     * and cause.
     * param message the detail message.
     * param cause the cause.
     */
    public IncorrectURL(String message, Throwable cause) {
        super(INCORRECT_URL, message, cause);
    }

    /**
     * Constructs an IncorrectURL exception with the specified detail message
     * and associated SAGA object.
     * param message the detail message.
     * param object the associated SAGA object.
     */
    public IncorrectURL(String message, SagaObject object) {
```

```
        super(INCORRECT_URL, message, object);
    }
}
```

3.1.4   BadParameter

```
package org.ogf.saga.error;

import org.ogf.saga.SagaObject;

/**
 * This exception indicates that one or more of the parameters of an
 * operation are ill-formed, invalid, out of bound, or otherwise not
 * usable.
 */
public class BadParameter extends Exception {

    private static final long serialVersionUID = 1L;

    /**
     * Constructs a BadParameter exception.
     */
    public BadParameter() {
        super(BAD_PARAMETER);
    }

    /**
     * Constructs a BadParameter exception with the specified detail
     * message.
     * param message the detail message.
     */
    public BadParameter(String message) {
        super(BAD_PARAMETER, message);
    }

    /**
     * Constructs a BadParameter exception with the specified cause.
     * param cause the cause.
     */
    public BadParameter(Throwable cause) {
        super(BAD_PARAMETER, cause);
    }

    /**
     * Constructs a BadParameter exception with the specified detail
     * message and cause.
     * param message the detail message.
     * param cause the cause.
```

```
     *
     */
    public BadParameter(String message, Throwable cause) {
        super(BAD_PARAMETER, message, cause);
    }

    /**
     * Constructs a BadParameter exception with the specified detail
     * message and associated SAGA object.
     * param message the detail message.
     * param object the associated SAGA object.
     */
    public BadParameter(String message, SagaObject object) {
        super(BAD_PARAMETER, message, object);
    }
}
```

3.1.5   AlreadyExists

```
package org.ogf.saga.error;

import org.ogf.saga.SagaObject;

/**
 * This exception indicate that an operation cannot succeed because the
 * entity to be created or registered already exists or is already registered.
 */
public class AlreadyExists extends Exception {

    private static final long serialVersionUID = 1L;

    /**
     * Constructs an AlreadyExists exception.
     */
    public AlreadyExists() {
        super(ALREADY_EXISTS);
    }

    /**
     * Constructs an AlreadyExists exception with the specified detail message.
     * param message the detail message.
     */
    public AlreadyExists(String message) {
        super(ALREADY_EXISTS, message);
    }

    /**
     * Constructs an AlreadyExists exception with the specified cause.
```

```
 * param cause the cause.
 */
public AlreadyExists(Throwable cause) {
    super(ALREADY_EXISTS, cause);
}

/**
 * Constructs an AlreadyExists exception with the specified detail message
 * and cause.
 * param message the detail message.
 * param cause the cause.
 */
public AlreadyExists(String message, Throwable cause) {
    super(ALREADY_EXISTS, message, cause);
}

/**
 * Constructs an AlreadyExists exception with the specified detail message
 * and associated SAGA object.
 * param message the detail message.
 * param object the associated SAGA object.
 */
public AlreadyExists(String message, SagaObject object) {
    super(ALREADY_EXISTS, message, object);
}
}
```

3.1.6   DoesNotExist

```
package org.ogf.saga.error;

import org.ogf.saga.SagaObject;

/**
 * This exception indicates that an operation cannot succeed because a
 * required entity is missing.
 */
public class DoesNotExist extends Exception {

    private static final long serialVersionUID = 1L;

    /**
     * Constructs a DoesNotExist exception.
     */
    public DoesNotExist() {
        super(DOES_NOT_EXIST);
    }
```

```
    /**
     * Constructs a DoesNotExist exception with the specified detail
     * message.
     * param message the detail message.
     */
    public DoesNotExist(String message) {
        super(DOES_NOT_EXIST, message);
    }

    /**
     * Constructs a DoesNotExist exception with the specified cause.
     * param cause the cause.
     */
    public DoesNotExist(Throwable cause) {
        super(DOES_NOT_EXIST, cause);
    }

    /**
     * Constructs a DoesNotExist exception with the specified detail
     * message and cause.
     * param message the detail message.
     * param cause the cause.
     *
     */
    public DoesNotExist(String message, Throwable cause) {
        super(DOES_NOT_EXIST, message, cause);
    }

    /**
     * Constructs a DoesNotExist exception with the specified detail
     * message and associated SAGA object.
     * param message the detail message.
     * param object the associated SAGA object.
     */
    public DoesNotExist(String message, SagaObject object) {
        super(DOES_NOT_EXIST, message, object);
    }
}
```

3.1.7   IncorrectState

```
package org.ogf.saga.error;

import org.ogf.saga.SagaObject;

/**
 * This exception indicates that the object on which a method is called is in a
 * state where that method cannot succeed.
```

```
 */
public class IncorrectState extends Exception {

    private static final long serialVersionUID = 1L;

    /**
     * Constructs an IncorrectState exception.
     */
    public IncorrectState() {
        super(INCORRECT_STATE);
    }

    /**
     * Constructs an IncorrectState exception with the specified detail message.
     * param message the detail message.
     */
    public IncorrectState(String message) {
        super(INCORRECT_STATE, message);
    }

    /**
     * Constructs an IncorrectState exception with the specified cause.
     * param cause the cause.
     */
    public IncorrectState(Throwable cause) {
        super(INCORRECT_STATE, cause);
    }

    /**
     * Constructs an IncorrectState exception with the specified detail message
     * and cause.
     * param message the detail message.
     * param cause the cause.
     */
    public IncorrectState(String message, Throwable cause) {
        super(INCORRECT_STATE, message, cause);
    }

    /**
     * Constructs an IncorrectState exception with the specified detail message
     * and associated SAGA object.
     * param message the detail message.
     * param object the associated SAGA object.
     */
    public IncorrectState(String message, SagaObject object) {
        super(INCORRECT_STATE, message, object);
    }
}
```

3.1.8   PermissionDenied

```
package org.ogf.saga.error;

import org.ogf.saga.SagaObject;

/**
 * This exception indicates that the identity used for the operation
 * did not have sufficient permissions to perform the operation successfully.
 */
public class PermissionDenied extends Exception {

    private static final long serialVersionUID = 1L;

    /**
     * Constructs a PermissionDenied exception.
     */
    public PermissionDenied() {
        super(PERMISSION_DENIED);
    }

    /**
     * Constructs a PermissionDenied exception with the specified detail
     * message.
     * param message the detail message.
     */
    public PermissionDenied(String message) {
        super(PERMISSION_DENIED, message);
    }

    /**
     * Constructs a PermissionDenied exception with the specified cause.
     * param cause the cause.
     */
    public PermissionDenied(Throwable cause) {
        super(PERMISSION_DENIED, cause);
    }

    /**
     * Constructs a PermissionDenied exception with the specified detail
     * message and cause.
     * param message the detail message.
     * param cause the cause.
     *
     */
    public PermissionDenied(String message, Throwable cause) {
        super(PERMISSION_DENIED, message, cause);
    }
```

```
    /**
     * Constructs a PermissionDenied exception with the specified detail
     * message and associated SAGA object.
     * param message the detail message.
     * param object the associated SAGA object.
     */
    public PermissionDenied(String message, SagaObject object) {
        super(PERMISSION_DENIED, message, object);
    }
}
```

3.1.9   AuthorizationFailed

```
package org.ogf.saga.error;

import org.ogf.saga.SagaObject;

/**
 * This exception indicates that a method fails because none of the available
 * session contexts could succesfully be used for authorization.
 */
public class AuthorizationFailed extends Exception {

    private static final long serialVersionUID = 1L;

    /**
     * Constructs an AuthorizationFailed exception.
     */
    public AuthorizationFailed() {
        super(AUTHORIZATION_FAILED);
    }

    /**
     * Constructs an AuthorizationFailed exception with the specified detail
     * message.
     * param message the detail message.
     */
    public AuthorizationFailed(String message) {
        super(AUTHORIZATION_FAILED, message);
    }

    /**
     * Constructs an AuthorizationFailed exception with the specified cause.
     * param cause the cause.
     */
    public AuthorizationFailed(Throwable cause) {
        super(AUTHORIZATION_FAILED, cause);
    }
```

```
    /**
     * Constructs an AuthorizationFailed exception with the specified detail
     * message and cause.
     * param message the detail message.
     * param cause the cause.
     */
    public AuthorizationFailed(String message, Throwable cause) {
        super(AUTHORIZATION_FAILED, message, cause);
    }

    /**
     * Constructs an AuthorizationFailed exception with the specified detail
     * message and associated SAGA object.
     * param message the detail message.
     * param object the associated SAGA object.
     */
    public AuthorizationFailed(String message, SagaObject object) {
        super(AUTHORIZATION_FAILED, message, object);
    }
}
```

3.1.10   AuthenticationFailed

```
package org.ogf.saga.error;

import org.ogf.saga.SagaObject;

/**
 * This exception indicates that a method fails because none of the available
 * session contexts could succesfully be used for authentication.
 */
public class AuthenticationFailed extends Exception {

    private static final long serialVersionUID = 1L;

    /**
     * Constructs an AuthenticationFailed exception.
     */
    public AuthenticationFailed() {
        super(AUTHENTICATION_FAILED);
    }

    /**
     * Constructs an AuthenticationFailed exception with the specified detail
     * message.
     * param message the detail message.
     */
```

```
    public AuthenticationFailed(String message) {
        super(AUTHENTICATION_FAILED, message);
    }

    /**
     * Constructs an AuthenticationFailed exception with the specified cause.
     * param cause the cause.
     */
    public AuthenticationFailed(Throwable cause) {
        super(AUTHENTICATION_FAILED, cause);
    }

    /**
     * Constructs an AuthenticationFailed exception with the specified detail
     * message and cause.
     * param message the detail message.
     * param cause the cause.
     */
    public AuthenticationFailed(String message, Throwable cause) {
        super(AUTHENTICATION_FAILED, message, cause);
    }

    /**
     * Constructs an AuthenticationFailed exception with the specified detail
     * message and associated SAGA object.
     * param message the detail message.
     * param object the associated SAGA object.
     */
    public AuthenticationFailed(String message, SagaObject object) {
        super(AUTHENTICATION_FAILED, message, object);
    }
}
```

3.1.11   Timeout

```
package org.ogf.saga.error;

import org.ogf.saga.SagaObject;

/**
 * This exception indicates that a remote operation did not complete
 * successfully because the network communication or the remote service
 * timed out.
 */
public class Timeout extends Exception {

    private static final long serialVersionUID = 1L;
```

```
/**
 * Constructs a Timeout exception.
 */
public Timeout() {
    super(TIMEOUT);
}

/**
 * Constructs a Timeout exception with the specified detail
 * message.
 * param message the detail message.
 */
public Timeout(String message) {
    super(TIMEOUT, message);
}

/**
 * Constructs a Timeout exception with the specified cause.
 * param cause the cause.
 */
public Timeout(Throwable cause) {
    super(TIMEOUT, cause);
}

/**
 * Constructs a Timeout exception with the specified detail
 * message and cause.
 * param message the detail message.
 * param cause the cause.
 *
 */
public Timeout(String message, Throwable cause) {
    super(TIMEOUT, message, cause);
}

/**
 * Constructs a Timeout exception with the specified detail
 * message and associated SAGA object.
 * param message the detail message.
 * param object the associated SAGA object.
 */
public Timeout(String message, SagaObject object) {
    super(TIMEOUT, message, object);
}
}
```

3.1.12   NoSuccess

```
package org.ogf.saga.error;

import org.ogf.saga.SagaObject;

/**
 * This exception indicates that an operation failed semantically.
 * This is the least specific exception in SAGA.
 */
public class NoSuccess extends Exception {

    private static final long serialVersionUID = 1L;

    /**
     * Constructs a NoSuccess exception.
     */
    public NoSuccess() {
        super(NO_SUCCESS);
    }

    /**
     * Constructs a NoSuccess exception with the specified detail
     * message.
     * param message the detail message.
     */
    public NoSuccess(String message) {
        super(NO_SUCCESS, message);
    }

    /**
     * Constructs a NoSuccess exception with the specified cause.
     * param cause the cause.
     */
    public NoSuccess(Throwable cause) {
        super(NO_SUCCESS, cause);
    }

    /**
     * Constructs a NoSuccess exception with the specified detail
     * message and cause.
     * param message the detail message.
     * param cause the cause.
     *
     */
    public NoSuccess(String message, Throwable cause) {
        super(NO_SUCCESS, message, cause);
    }
```

```
    /**
     * Constructs a NoSuccess exception with the specified detail
     * message and associated SAGA object.
     * param message the detail message.
     * param object the associated SAGA object.
     */
    public NoSuccess(String message, SagaObject object) {
        super(NO_SUCCESS, message, object);
    }
}
```

3.2 SAGA Base Object

The SAGA object interface provides methods which are essential for all SAGA
objects. It provides a unique ID which helps maintain a list of SAGA objects
at the application level as well as allowing for inspection of objects type and its
associated session.

The object id MUST be formatted as UUID, as standardized by the Open
Software Foundation (OSF) as part of the Distributed Computing Environment
(DCE). The UUID format is also described in the IETF RFC-4122 [6].

3.2.1 SagaObject

```
package org.ogf.saga;

import org.ogf.saga.session.Session;
import org.ogf.saga.error.DoesNotExist;

/**
 * This is the base for all SAGA objects.
 * Deviation from the SAGA specs: we don't want to call this
 * "Object" because that might cause some confusion in Java.
 * All SAGA objects must support the clone() method, so this interface
 * extends Cloneable.
 */
public interface SagaObject extends Cloneable {

    /** Timeout constant: wait forever. */
    public static final float WAIT_FOREVER = -1.0F;

    /** Timeout constant: don't wait. */
    public static final float NO_WAIT = 0.0F;

    /**
     * Returns a shallow copy of the session from which this object was
     * created.
     * return the session.
     * exception DoesNotExist is thrown when this method is called on objects
     *     that do not have a session attached.
     */
    public Session getSession() throws DoesNotExist;

    /**
     * Returns the object id of this SAGA object.
     * Note: java.util.UUID could be used for this.
     * See Sun's comments on UUID generation.
     * return the object id.
```

```
 */
public String getId();

/**
 * Copies the Saga object.
 * return the clone.
 * throws CloneNotSupportedException when the clone method is not supported.
 */
public Object clone() throws CloneNotSupportedException;

/**
 * Returns the object type. The SAGA application could use
 * introspection, but would then get class names that are
 * specific for a particular SAGA implementation.
 */
public ObjectType getType();
}
```

3.2.2 ObjectType

```
package org.ogf.saga;

/**
 * Enumerates the different object types in SAGA.
 */
public enum ObjectType {

    UNKNOWN (-1),
    EXCEPTION (1),
    URL(2),
    BUFFER(3),
    SESSION (4),
    CONTEXT (5),
    TASK (6),
    TASKCONTAINER (7),
    METRIC (8),
    NSENTRY (9),
    NSDIRECTORY (10),
    IOVEC (11),
    FILE (12),
    DIRECTORY (13),
    LOGICALFILE (14),
    LOGICALDIRECTORY (15),
    JOBDESCRIPTION (16),
    JOBSERVICE (17),
    JOB (18),
    JOBSELF (19),
    STREAMSERVICE (20),
```

```
    STREAM (21),
    PARAMETER (22),
    RPC (23),

    // Added object type for Java bindings.
    FILEINPUTSTREAM(24),
    FILEOUTPUTSTREAM(25);


    private int value;

    ObjectType(int value) {
        this.value = value;
    }

    /**
     * Returns the integer value of this enumeration literal.
     * return the integer value.
     */
    public int getValue() {
        return value;
    }
}
```

3.3   SAGA URL Class

In many places in the SAGA API, URLs are used to reference remote entities.
In order to

- simplify the construction and the parsing of URLs on application level,

- allow for sanity checks within and outside the SAGA implementation,

- simplify and unify the signatures of SAGA calls which accept URLs,

a SAGA URL class is used. This class provides means to set and access the
various elements of a URL. The class parses the URL in conformance to RFC-
3986 [1].

In respect to the URL problem (stated in GFD.90, Section 2.11), the class pro-
vides the method `translate (in string scheme)`, which allows to translate
a URL from one scheme to another – with all the limitations mentioned in
GFD.90, Section 2.11.

3.3.1   URL

```
package org.ogf.saga;

import java.net.URISyntaxException;
import java.net.URI;

import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;

/**
 * URL class as specified by SAGA. The java.net.URL class is not usable
 * because of all kinds of side-effects.
 * TODO: provide factory with this as a default implementation???
 */
public class URL {
    private URI u;

    /**
     * Constructs an URL from the specified string.
     * param url the string.
     * exception BadParameter is thrown when there is a syntax error
     *      in the parameter.
     */
    public URL(String url) throws NotImplemented, BadParameter, NoSuccess {
```

```
        try {
            u = new URI(url);
        } catch(URISyntaxException e) {
            throw new BadParameter("syntax error in url", e);
        }
    }

    private URL(URI u) {
        this.u = u;
    }

    /**
     * Replaces the current value of the URL with the specified value.
     * param url the string.
     * exception BadParameter is thrown when there is a syntax error
     *     in the parameter.
     */
    public void setURL(String url) throws NotImplemented, BadParameter {
        try {
            u = new URI(url);
        } catch(URISyntaxException e) {
            throw new BadParameter("syntax error in url", e);
        }
    }

    /**
     * Returns this URL as a string.
     * return the string.
     */
    public String getURL() throws NotImplemented {
        return toString();
    }

    /**
     * Returns the fragment part of this URL.
     * return the fragment.
     */
    public String getFragment() throws NotImplemented {
        return u.getFragment();
    }

    /**
     * Sets the fragment part of this URL to the specified parameter.
     * param fragment the fragment.
     * exception BadParameter is thrown when there is a syntax error in the
     * parameter.
     */
    public void setFragment(String fragment) throws NotImplemented,
            BadParameter {
        try {
```

```
            u = new URI(u.getScheme(), u.getUserInfo(), u.getHost(),
                    u.getPort(), u.getPath(), u.getQuery(), fragment);
        } catch(URISyntaxException e) {
            throw new BadParameter("syntax error in fragment", e);
        }
    }

    /**
     * Returns the host part of this URL.
     * return the host.
     */
    public String getHost() throws NotImplemented {
        return u.getHost();
    }

    /**
     * Sets the host part of this URL to the specified parameter.
     * param host the host.
     * exception BadParameter is thrown when there is a syntax error in the
     * parameter.
     */
    public void setHost(String host) throws NotImplemented, BadParameter {
        try {
            u = new URI(u.getScheme(), u.getUserInfo(), host,
                    u.getPort(), u.getPath(), u.getQuery(), u.getFragment());
        } catch(URISyntaxException e) {
            throw new BadParameter("syntax error in host", e);
        }
    }

    /**
     * Returns the path part of this URL.
     * return the path.
     */
    public String getPath() throws NotImplemented {
        return u.getPath();
    }

    /**
     * Sets the path part of this URL to the specified parameter.
     * param path the path.
     * exception BadParameter is thrown when there is a syntax error in the
     * path.
     */
    public void setPath(String path) throws NotImplemented, BadParameter {
        try {
            u = new URI(u.getScheme(), u.getUserInfo(), u.getHost(),
                    u.getPort(), path, u.getQuery(), u.getFragment());
        } catch(URISyntaxException e) {
            throw new BadParameter("syntax error in host", e);
```

```
        }
    }

    /**
     * Returns the port number of this URL.
     * return the port number.
     */
    public int getPort() throws NotImplemented {
        return u.getPort();
    }

    /**
     * Sets the port number of this URL to the specified parameter.
     * param port the port number.
     * exception BadParameter is thrown when there is a syntax error in the
     * parameter. (???)
     */
    public void setPort(int port) throws NotImplemented, BadParameter {
        try {
            u = new URI(u.getScheme(), u.getUserInfo(), u.getHost(),
                    port, u.getPath(), u.getQuery(), u.getFragment());
        } catch(URISyntaxException e) {
            throw new BadParameter("syntax error in port", e);      // ???
        }
    }

    /**
     * Returns the query part from this URL.
     * return the query.
     */
    public String getQuery() throws NotImplemented {
        return u.getQuery();
    }

    /**
     * Sets the query part of this URL to the specified parameter.
     * param query the query.
     * exception BadParameter is thrown when there is a syntax error in the
     * parameter.
     */
    public void setQuery(String query) throws NotImplemented, BadParameter {
        try {
            u = new URI(u.getScheme(), u.getUserInfo(), u.getHost(),
                    u.getPort(), u.getPath(), query, u.getFragment());
        } catch(URISyntaxException e) {
            throw new BadParameter("syntax error in query", e);
        }
    }

    /**
```

```
 * Returns the scheme part from this URL.
 * return the scheme.
 */
public String getScheme() throws NotImplemented {
    return u.getScheme();
}

/**
 * Sets the scheme part of this URL to the specified parameter.
 * param scheme the scheme.
 * exception BadParameter is thrown when there is a syntax error in the
 * parameter.
 */
public void setScheme(String scheme) throws NotImplemented, BadParameter {
    try {
        u = new URI(scheme, u.getUserInfo(), u.getHost(),
                u.getPort(), u.getPath(), u.getQuery(), u.getFragment());
    } catch(URISyntaxException e) {
        throw new BadParameter("syntax error in scheme", e);
    }
}

/**
 * Returns the userinfo part from this URL.
 * return the userinfo.
 */
public String getUserInfo() throws NotImplemented {
    return u.getUserInfo();
}

/**
 * Sets the user info part of this URL to the specified parameter.
 * param userInfo the userinfo.
 * exception BadParameter is thrown when there is a syntax error in the
 * parameter.
 */
public void setUserInfo(String userInfo) throws NotImplemented,
        BadParameter {
    try {
        u = new URI(u.getScheme(), userInfo, u.getHost(),
                u.getPort(), u.getPath(), u.getQuery(), u.getFragment());
    } catch(URISyntaxException e) {
        throw new BadParameter("syntax error in query", e);
    }
}

/**
 * Returns a new URL with the scheme part replaced.
 * param scheme the new scheme.
 * return the new URL.
```

```
 * exception BadParameter is thrown when there is a syntax error in the
 *     new URL.
 */
public URL translate(String scheme) throws NotImplemented, BadParameter,
        NoSuccess {
    try {
        URI url = new URI(scheme, u.getUserInfo(), u.getHost(),
                u.getPort(), u.getPath(), u.getQuery(), u.getFragment());
        // Not quite correct: the SAGA specs say that NoSuccess should be
        // thrown when the scheme is not supported. How to check this
        // here ???
        return new URL(url);
    } catch(URISyntaxException e) {
        throw new BadParameter("syntax error in scheme", e);
    }
}

public int hashCode() {
    return u.hashCode();
}

public boolean equals(Object o) {
    if (o == null) {
        return false;
    }
    if (! (o instanceof URL)) {
        return false;
    }
    URL other = (URL) o;
    return u.equals(other.u);
}

public String toString() {
    return u.toString();
}

/**
 * See {link java.net.URI#resolve(java.net.URI)}.
 * param url the url to resolve with respect to this one.
 * return the resolved url.
 */
public URL resolve(URL url) throws NoSuccess {
    URI uri = u.resolve(url.u);
    if (uri == url.u) {
        return url;
    }

    return new URL(uri);
}
```

```
/**
 * See {link java.net.URI#isAbsolute()}.
 * return whether this URL is an absolute URL.
 */
public boolean isAbsolute() {
    return u.isAbsolute();
}

/**
 * See {link java.net.URI#normalize()}.
 * return a normalized URL.
 */
public URL normalize() {
    URI uri = u.normalize();
    if (uri == u) {
        return this;
    }
    return new URL(uri);
}
}
```

3.4   SAGA I/O Buffer

The SAGA API includes a number of calls which perform byte-level I/O opera-
tions, e.g. `read()`/`write()` on files and streams, and `call()` on rpc instances.
Future SAGA API extensions are expected to increase the number of I/O meth-
ods. The `saga::buffer` class encapsulates a sequence of bytes to be used for
such I/O operations – that allows for uniform I/O syntax and semantics over
the various SAGA API packages.

The class is designed to be a simple container containing one single element (the
opaque data). The data can either be allocated and maintained in application
memory, or can be allocated and maintained by the SAGA implementation.
The latter is the default, and applies when no data and no size are specified on
buffer construction.

For example, an application that has data memory already allocated and filled,
can create and use a buffer by calling

```
// create buffer with application memory
char data[1000];
saga::buffer b (data, 1000);
```

The same also works when used with the respective I/O operations:

```
// write to a file using a buffer with application  memory
char data[1000] = ...;
file.write (saga::buffer (data, 1000));
```

Another application, which wants to leave the buffer memory management to
the SAGA implementation, can use a second constructor, which causes the
implementation to allocate memory on the fly:

```
// create empty, implementation managed buffer
saga::buffer b;  // no data nor size given!

// read 100 byte from file into buffer
file.read (b, 100);

// get memory from SAGA
const char * data = b.get_data ();

// or use data directly
std::cout << "found: " << b.get_data () << std::endl;
```

Finally, an application can leave memory management to the implementation,
as above, but can specify how much memory should be allocated by the SAGA
implementation:

```
// create an implementation managed buffer of 100 byte
saga::buffer b (100);

// get memory from SAGA
const char * data = b.get_data ();

// fill the buffer
memcopy (data, source, b.get_size ());

// use data for write
file.write (b);
```

Application-managed memory MUST NOT be re- or de-allocated by the SAGA implementation, and implementation-managed memory MUST NOT be re- or de-allocated by the application. However, an application CAN change the *content* of implementation managed memory, and vice versa.

Also, a buffer's contents MUST NOT be changed by the application while it is in use, i.e. while any I/O operation on that buffer is ongoing. For asynchronous operations, an I/O operation is considered ongoing if the associated `saga::task` instance is not in a final state.

If a buffer is too small (i.e. more data are available for a read, or more data are required for a write), only the available data are used, and an error is returned appropriately. If a buffer is too large (i.e. read is not able to fill the buffer completely, or write does not need the complete buffer), the remainder of the buffer data MUST be silently ignored (i.e. not changed, and not set to zero). The error reporting mechanisms as listed for the specific I/O methods apply.

Implementation-managed memory is released when the buffer is destroyed, (either explicitly by calling `close()`, or implicitly by going out of scope). It MAY be re-allocated, and reset to zero, if the application calls `set_size()`.

Application-managed memory is released by the application. In order to simplify memory management, language bindings (in particular for non-garbage-collecting languages) MAY allow to register a callback on buffer creation which is called on buffer destruction, and which can be used to de-allocate the buffer memory in a timely manner. The `saga::callback` class SHOULD be used for that callback – those language bindings SHOULD thus define the buffer to be `monitorable`, i.e. it should implement the `saga::monitorable` interface. After the callback's invocation, the buffer MUST NOT be used by the implementation anymore.

When calling `set_data()` for application-managed buffers, the implementation MAY copy the data internally, or MAY use the given data pointer as is. The application SHOULD thus not change the data while an I/O operation is in progress, and only consider the data pointer to be unused after another `set_data()` has been called, or the buffer instance was destroyed.

Note that these conventions on memory management allow for zero-copy SAGA implementations, and also allow to reuse buffer instances for multiple I/O operations, which makes, for example, the implementation of pipes and filters very simple.

The buffer class is designed to be inherited by application-level I/O buffers, which may, for example, add custom data getter and setter methods (e.g. `set_jpeg()` and `get_jpeg()`. Such derived buffer classes can thus add both data formats and data models transparently on top of SAGA I/O. For developers who program applications for a specific community it seems advisable to standardize both data format and data model, and possibly to standardize derived SAGA buffers – that work is, at the moment, out of scope for SAGA. The SAGA API MAY, however, specify such derived buffer classes in later versions, or in future extensions of the API.

A buffer does not belong to a session, and a buffer object instance can thus be used in multiple sessions, for I/O operations on different SAGA objects.

Note that even if a buffer size is given, the `len_in` parameter to the SAGA I/O operations supersedes the buffer size. If the buffer is too small, a `'BadParameter'` exception will be thrown on these operations. If `len_in` is ommitted and the buffer size is not known, a `'BadParameter'` exception is also thrown.

Note also that the `len_out` parameter of the SAGA I/O operations has not necessarily the same value as the buffer size, obtained with `buffer.get_size()`. A read may read only a part of the requested data, and a write may have written only a part of the buffer. That is not an error, as is described in the notes for the respective I/O operations.

SAGA language bindings may want to define a `const`-version of the buffer, in order to allow for safe implementations. A non-const buffer SHOULD then inherit the `const` buffer class, and add the appropriate constructor and setter methods. The same holds for SAGA classes which inherit from the `buffer`.

Also, language bindings MAY allow buffer constructors with optional `size` parameter, if the size of the given data is implicitly known. For example, the C++ bindings MAY allow an buffer constructor `buffer (std::string s)`. The same holds for SAGA classes that inherit from the `buffer`.

In Java, arrays have a size that can be examined at run-time, so the buffer creation methods in the Java language bindings either specify a `size`, in which case the buffer is implementation-managed, or a byte array, in which case the buffer is application-managed.

3.4.1 Buffer

```
package org.ogf.saga.buffer;

import org.ogf.saga.SagaObject;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.DoesNotExist;
import org.ogf.saga.error.IncorrectState;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;

/**
 * Encapsulates a sequence of bytes to be used for I/O operations.
 * ??? Should we just use java.nio.ByteBuffer instead ??? Probably yes.
 * An implementation could encapsulate a java.nio.ByteBuffer in a Buffer,
 * too.
 * ???Java mostly uses <code>int</code> for buffer sizes. Should we use
 * <code>long</code>???
 */
public interface Buffer extends SagaObject {

    /**
     * Sets the size of the buffer. This method is semantically equivalent
     * to re-creating it with the specified size.
     * This makes the buffer implementation-allocated, unless size = -1,
     * in which case it becomes implementation-managed.
     * Spec inconsistency: this method should also throw NoSuccess, as the
     * constructor can throw this, and this method is semantically equivalent
     * to destruct and then call constructor.
     * param size the size.
     */
    public void setSize(int size)
        throws NotImplemented, BadParameter, IncorrectState, NoSuccess;

    /**
     * Sets the size of the buffer. This method is semantically equivalent
     * to re-creating it.
     * This method makes the buffer implementation-managed.
     * Spec inconsistency: this method should also throw NoSuccess, as the
     * constructor can throw this, and this method is semantically equivalent
     * to destruct and then call constructor.
     */
    public void setSize()
        throws NotImplemented, BadParameter, IncorrectState, NoSuccess;

    /**
     * Retrieves the current value of the buffer size.
     * return the size.
     */
```

```
    public int getSize()
        throws NotImplemented, IncorrectState;

    /**
     * Sets the buffer data. Makes the buffer application-managed.
     * Deviation from the SAGA specs: the size is implicit in the byte
     * array. Calling this method implies: user-allocated data.
     * Spec inconsistency: this method should also throw NoSuccess, as the
     * constructor can throw this, and this method is semantically equivalent
     * to destruct and then call constructor.
     * param data the data.
     */
    public void setData(byte[] data)
        throws NotImplemented, BadParameter, IncorrectState, NoSuccess;

    /**
     * Retrieves the buffer data.
     * return the data.
     * exception DoesNotExist is thrown when the buffer was created with
     *      size -1, and no I/O operation has been done on it yet.
     */
    public byte[] getData()
        throws NotImplemented, DoesNotExist, IncorrectState;

    /**
     * Non-blocking close of the buffer object.
     */
    public void close()
        throws NotImplemented;

    /**
     * Closes the buffer object.
     * param timeoutInSeconds the timeout in seconds.
     */
    public void close(float timeoutInSeconds)
        throws NotImplemented;
}
```

3.4.2   BufferFactory

```
package org.ogf.saga.buffer;

import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;

/**
```

```
 * Factory for creating buffers.
 */
public abstract class BufferFactory {

    private static BufferFactory factory;

    private synchronized static void initFactory() throws NotImplemented {
        if (factory == null) {
            factory = ImplementationBootstrapLoader.createBufferFactory();
        }
    }

    /**
     * Creates a buffer. To be provided by an implementation.
     * param data the storage.
     * return the buffer.
     */
    protected abstract Buffer doCreateBuffer(byte[] data)
        throws NotImplemented, BadParameter, NoSuccess;

    /**
     * Creates a buffer. To be provided by an implementation.
     * return the buffer.
     */
    protected abstract Buffer doCreateBuffer()
        throws NotImplemented, BadParameter, NoSuccess;

    /**
     * Creates a buffer. To be provided by an implementation.
     * param size the size of the buffer.
     * return the buffer.
     */
    protected abstract Buffer doCreateBuffer(int size)
        throws NotImplemented, BadParameter, NoSuccess;

    /**
     * Creates a (application-allocated) buffer. The size is implicit in
     * the size of the specified array.
     * param data the storage.
     * return the buffer.
     * throws NoSuccess
     */
    public static Buffer createBuffer(byte[] data)
        throws NotImplemented, BadParameter, NoSuccess {
        initFactory();
        return factory.doCreateBuffer(data);
    }

    /**
     * Creates a (implementation-managed and implementation-allocated) buffer
```

```
 * of the specified size.
 * param size the size.
 * return the buffer.
 */
public static Buffer createBuffer(int size)
    throws NotImplemented, BadParameter, NoSuccess {
    initFactory();
    return factory.doCreateBuffer(size);
}


/**
 * Creates a (implementation-managed) buffer.
 * return the buffer.
 * throws NoSuccess
 */
public static Buffer createBuffer()
    throws NotImplemented, BadParameter, NoSuccess {
    initFactory();
    return factory.doCreateBuffer();
}
}
```

3.5   SAGA Session Management

The session object provides the functionality of a session, which isolates independent sets of SAGA objects from each other. Sessions also support the management of security information (see `saga::context` in Section 3.6).

3.5.1   Session

```
package org.ogf.saga.session;

import org.ogf.saga.SagaObject;
import org.ogf.saga.context.Context;
import org.ogf.saga.error.DoesNotExist;
import org.ogf.saga.error.NotImplemented;

/**
 * A session isolates independent sets of SAGA objects from each other,
 * and support management of security contexts.
 */
public interface Session extends SagaObject {
    /**
     * Attaches a deep copy of the specified security context to the session.
     * param context the context to be added.
     */
    public void addContext(Context context) throws NotImplemented;

    /**
     * Detaches the specified security context from the session.
     * param context the context to be removed.
     * exception DoesNotExist is thrown when the session does not
     *     contain the specified context.
     */
    public void removeContext(Context context) throws NotImplemented,
            DoesNotExist;

    /**
     * Retrieves all contexts attached to the session.
     * An empty array is returned if no context is attached.
     * return a list of contexts.
     */
    public Context[] listContexts() throws NotImplemented;

    /**
     * Closes a SAGA session. Deviation from the SAGA API, which does not
     * have this method. However, middleware may for instance have threads
     * which may need to be terminated, or the application will hang.
     * This may not be the right place for it, but there is no other place ...
```

```
    */
   public void close() throws NotImplemented;

   /**
    * Closes a SAGA session. Deviation from the SAGA API, which does not
    * have this method. However, middleware may for instance have threads
    * which may need to be terminated, or the application will hang.
    * param timeoutInSeconds the timeout in seconds.
    */
   public void close(float timeoutInSeconds) throws NotImplemented;
}
```

3.5.2   SessionFactory

```
package org.ogf.saga.session;

import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;

/**
 * Factory for creating sessions.
 */
public abstract class SessionFactory {

   private static SessionFactory factory;

   private synchronized static void initFactory() {
       if(factory == null) {
           factory = ImplementationBootstrapLoader.createSessionFactory();
       }
   }

   /**
    * Creates a session. To be provided by an implementation.
    * param defaults when set, the default session is
    * returned, with all the default contexts. Later modifications to this
    * session are reflected in the default session.
    * return the session.
    */
   protected abstract Session doCreateSession(boolean defaults)
           throws NotImplemented, NoSuccess;

   /**
    * Creates a session.
    * param defaults when set, the default session is
    * returned, with all the default contexts. Later modifications to this
    * session are reflected in the default session.
```

```
 * return the session.
 */
public static Session createSession(boolean defaults)
        throws NotImplemented, NoSuccess {
    initFactory();
    return factory.doCreateSession(defaults);
}

/**
 * Returns the default session, with all the default contexts.
 * Later modifications to this session are reflected in the default session.
 * return the session.
 */
public static Session createSession()
        throws NotImplemented, NoSuccess {
    return createSession(true);
}
}
```

3.6   SAGA Context Management

The `saga::context` class provides the functionality of a security information
container. A context gets created, and attached to a session handle. As such
it is available to all objects instantiated in that session. Multiple contexts can
co-exist in one session – it is up to the implementation to choose the correct
context for a specific method call. Also, a single `saga::context` instance can
be shared between multiple sessions. SAGA objects created from other SAGA
objects inherit its session and thus also its context(s).

A context has a set of attributes which can be set/get via the SAGA attributes
interface. Exactly which attributes a context actually evaluates, depends upon
its type (see documentation to the `set_defaults()` method.)

An implementation CAN implement multiple types of contexts. The implemen-
tation MUST document which context types it supports, and which values to
the `Type` attribute are used to identify these context types. Also, the implemen-
tation MUST document which default values it supports for the various context
types, and which attributes need to be or can be set by the application.

The lifetime of `saga::context` instances is defined by the lifetime of those
`saga::session` instances the contexts are associated with, and of those SAGA
objects which have been created in these sessions. For detailed information
about lifetime management, see GFD.90.

For application level Authorization (e.g. for streams, monitoring, steering), con-
texts are used to inform the application about the requestor's identity. These
contexts represent the security information that has been used to initiate the
connection to the SAGA application. To support that mechanism, a num-
ber of specific attributes are available, as specified below. They are named
`"Remote<attribute>"`. An implementation MUST at least set the `Type` at-
tribute for such contexts, and SHOULD provide as many attribute values as
possible.

Note that `UserID`s SHOULD be formatted so that they can be used as user
identifiers in the SAGA permission model – see Section 3.7 for details.

3.6.1   Context

```
package org.ogf.saga.context;

import org.ogf.saga.SagaObject;
import org.ogf.saga.attributes.Attributes;
import org.ogf.saga.error.IncorrectState;
import org.ogf.saga.error.NoSuccess;
```

```
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.Timeout;

/**
 * A <code>Context</code> provides the functionality of a security information
 * container.
 */
public interface Context extends SagaObject, Attributes {
    /** Attribute name: type of context. */
    public static final String TYPE = "Type";

    /** Attribute name: server which manages the context. */
    public static final String SERVER = "Server";

    /** Attribute name: Location of certificates and CA signatures. */
    public static final String CERTREPOSITORY = "CertRepository";

    /** Attribute name: Location of an existing certificate proxy to be used. */
    public static final String USERPROXY = "UserProxy";

    /** Attribute name: Location of a user certificate to be used. */
    public static final String USERCERT = "UserCert";

    /** Attribute name: Location of a user key to use. */
    public static final String USERKEY = "UserKey";

    /** Attribute name: User ID or user name to use. */
    public static final String USERID = "UserID";

    /** Attribute name: Password to use. */
    public static final String USERPASS = "UserPass";

    /** Attribute name: The VO the context belongs to. */
    public static final String USERVO = "UserVO";

    /** Attribute name: Time up to which this context is valid. */
    public static final String LIFETIME = "LifeTime";

    /**
     * Attribute name: User ID for a remote user, who is identified by this
     * context. (ReadOnly)
     */
    public static final String REMOTEID = "RemoteID";

    /**
     * Attribute name: The hostname where the connection originates which is
     * identified by this context. (ReadOnly)
     */
    public static final String REMOTEHOST = "RemoteHost";
```

```
    /**
     * Attribute name: the port used for the connection which is identified
     * by this context. (ReadOnly)
     */
    public static final String REMOTEPORT = "RemotePort";

    /**
     * Sets default attribute values for this context type, based on all
     * non-empty attributes.
     */
    public void setDefaults()
        throws NotImplemented, IncorrectState, Timeout, NoSuccess;
}
```

3.6.2   ContextFactory

```
package org.ogf.saga.context;

import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.IncorrectState;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.Timeout;



/**
 * Factory for objects in the saga.context package.
 */
public abstract class ContextFactory {

    private static ContextFactory factory;

    /**
     * Constructs a security context. To be provided by the implementation.
     * param type when set to a non-empty string, {@link Context#setDefaults()}
     * is called.
     * return the security context.
     */
    protected abstract Context doCreateContext(String type)
            throws NotImplemented, IncorrectState, Timeout, NoSuccess;

    private synchronized static void initFactory() {
        if(factory == null) {
            factory = ImplementationBootstrapLoader.createContextFactory();
        }
    }
```

```
    /**
     * Constructs a security context.
     * param type when set to a non-empty string, {@link Context#setDefaults()}
     * is called.
     * return the security context.
     */
    public static Context createContext(String type) throws NotImplemented,
            IncorrectState, Timeout, NoSuccess {
        initFactory();
        return factory.doCreateContext(type);
    }

    /**
     * Constructs a security context.
     * return the security context.
     */
    public static Context createContext() throws NotImplemented,
            IncorrectState, Timeout, NoSuccess {
        return createContext("");
    }
}
```

3.7   SAGA Permission Model

A number of SAGA use cases imply the ability of applications to allow or deny specific operations on SAGA objects or grid entities, such as files, streams, or monitorables. This package provides a generic interface to query and set such permissions, for (a) everybody, (b) individual users, and (c) groups of users.

Objects implementing this interface maintain a set of permissions for each object instance, for a set of IDs. These permissions can be queried, and, in many situations, set. The SAGA specification defines which permissions are available on a SAGA object, and which operations are expected to respect these permissions.

3.7.1   Permissions

```
package org.ogf.saga.permissions;

import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;
import org.ogf.saga.task.Async;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;

/**
 * This interface describes methods to query and set permissions.
 */
public interface Permissions extends Async {

    /**
     * Allows the specified permissions for the specified id.
     * An id of "*" enables the permissions for all.
     * param id the id.
     * param permissions the permissions to enable.
     */
    public void permissionsAllow(String id, int permissions)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, BadParameter, Timeout, NoSuccess;

    /**
     * Denies the specified permissions for the specified id.
     * An id of "*" disables the permissions for all.
     * param id the id.
```

```
 * param permissions the permissions to disable.
 */
public void permissionsDeny(String id, int permissions)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, Timeout, NoSuccess;


/**
 * Determines if the specified permissions are enabled for the
 * specified id.
 * An id of "*" queries the permissions for all.
 * param id the id.
 * param permissions the permissions to query.
 * return <code>true</code> if the specified permissions are enabled
 *     for the specified id.
 */
public boolean permissionsCheck(String id, int permissions)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, Timeout, NoSuccess;


/**
 * Gets the owner id of the entity.
 * return the id of the owner.
 */
public String getOwner()
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, Timeout, NoSuccess;


/**
 * Gets the group id of the entity.
 * return the id of the group.
 */
public String getGroup()
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, Timeout, NoSuccess;


/**
 * Creates a task that enables the specified permissions for the
 * specified id.
 * An id of "*" enables the permissions for all.
 * param mode determines the initial state of the task.
 * param id the id.
 * param permissions the permissions to enable.
 * return the task object.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task permissionsAllow(TaskMode mode, String id,
        int permissions)
    throws NotImplemented;
```

```
    /**
     * Creates a task that disables the specified permissions for the
     * specified id.
     * An id of "*" disables the permissions for all.
     * param mode determines the initial state of the task.
     * param id the id.
     * param permissions the permissions to disable.
     * return the task object.
     * exception NotImplemented is thrown when the task version of this
     *      method is not implemented.
     */
    public Task permissionsDeny(TaskMode mode, String id,
            int permissions)
        throws NotImplemented;

    /**
     * Creates a task that determines if the specified permissions are
     * enabled for the specified id.
     * An id of "*" queries the permissions for all.
     * param mode determines the initial state of the task.
     * param id the id.
     * param permissions the permissions to query.
     * return the task object.
     * exception NotImplemented is thrown when the task version of this
     *      method is not implemented.
     */
    public Task<Boolean> permissionsCheck(TaskMode mode, String id,
            int permissions)
        throws NotImplemented;

    /**
     * Creates a task that obtains the owner id of the entity.
     * param mode determines the initial state of the task.
     * return the task object.
     * exception NotImplemented is thrown when the task version of this
     *      method is not implemented.
     */
    public Task<String> getOwner(TaskMode mode)
        throws NotImplemented;

    /**
     * Creates a task that obtains the group id of the entity.
     * param mode determines the initial state of the task.
     * return the task object.
     * exception NotImplemented is thrown when the task version of this
     *      method is not implemented.
     */
    public Task<String> getGroup(TaskMode mode)
        throws NotImplemented;
}
```

3.7.2 Permission

```
package org.ogf.saga.permissions;

/**
 * Enumerates all permission values.
 * These flags are meant to be or-ed together, resulting in an integer,
 * so methods are added to test for presence and or-ing.
 */
public enum Permission {
    UNKNOWN(-1),
    NONE(0),
    QUERY(1),
    READ(2),
    WRITE(4),
    EXEC(8),
    OWNER(16),
    ALL(31);

    private int value;

    Permission(int value) {
        this.value = value;
    }

    /**
     * Returns the integer value of this permission.
     * return the integer value.
     */
    public int getValue() {
        return value;
    }

    /**
     * Returns the result of or-ing this flag into an integer.
     * param val the value to OR this enumeration value into.
     * return the result of or-ing this flag into the integer parameter.
     */
    public int or(int val) {
        return val | value;
    }

    /**
     * Returns the result of or-ing this flag into another.
     * param val the value to OR this enumeration value into.
     * return the result of or-ing this flag into the integer parameter.
     */
    public int or(Permission val) {
        return val.value | value;
```

```
    }

    /**
     * Tests for the presence of this flag in the specified value.
     * param val the value.
     * return <code>true</code> if this flag is present.
     */
    public boolean isSet(int val) {
        if (value == val) {
            // Also tests for 0 (NONE) which is assumed to be set only when
            // no other values are set.
            return true;
        }
        return (val & value) != 0;
    }
}
```

3.8   SAGA Attribute Model

There are various places in the SAGA API where attributes need to be associated with objects, for instance for job descriptions and metrics. The `attributes` interface provides a common interface for storing and retrieving attributes.

Objects implementing this interface maintain a set of attributes. These attributes can be considered as a set of key-value pairs attached to the object. The key-value pairs are string based for now, but might cover other value types in later versions of the SAGA API specification.

3.8.1   Attributes

```
package org.ogf.saga.attributes;

import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.DoesNotExist;
import org.ogf.saga.error.IncorrectState;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;

/**
 * Provides a uniform paradigm to set and query parameters and properties
 * of SAGA objects.
 * Attributes map a key to a value.
 */
public interface Attributes {

    // Attribute types
    public static final String STRING = "String";
    public static final String INT = "Int";
    public static final String ENUM = "Enum";
    public static final String FLOAT = "Float";
    public static final String BOOL = "Bool";
    public static final String TIME = "Time";
    // For "trigger" metrics:
    public static final String TRIGGER = "Trigger";

    // Boolean values:
    public static final String TRUE = "True";
    public static final String FALSE = "False";

    /**
```

```
 * Sets an attribute to a value.
 * param key the attribute key.
 * param value value to set the attribute to.
 */
public void setAttribute(String key, String value)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectState, BadParameter,
        DoesNotExist, Timeout, NoSuccess;

/**
 * Gets the value of an attribute.
 * param key the attribute key.
 * return the value of this attribute.
 */
public String getAttribute(String key)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectState, DoesNotExist, Timeout, NoSuccess;

/**
 * Sets an attribute to an array of values.
 * param key the attribute key.
 * param values values to set the attribute to.
 */
public void setVectorAttribute(String key, String[] values)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectState, BadParameter,
        DoesNotExist, Timeout, NoSuccess;

/**
 * Gets the array of values associated with an attribute.
 * param key the attribute key.
 * return the values of this attribute, or <code>null</code>.
 */
public String[] getVectorAttribute(String key)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectState, DoesNotExist, Timeout, NoSuccess;

/**
 * Removes an attribute.
 * param key the attribute key.
 */
public void removeAttribute(String key)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, DoesNotExist, Timeout, NoSuccess;

/**
 * Gets the list of attribute keys.
 * return the list of attribute keys.
 */
public String[] listAttributes()
```

```
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, Timeout, NoSuccess;


    /**
     * Finds matching attributes.
     * param patterns the search patterns.
     * return the list of matching attribute keys.
     */
    public String[] findAttributes(String... patterns)
        throws NotImplemented, BadParameter, AuthenticationFailed,
            AuthorizationFailed, PermissionDenied, Timeout, NoSuccess;


    /**
     * Checks the attribute for being read-only.
     * param key the attribute key.
     * return <code>true</code> if the attribute exists and is read-only.
     */
    public boolean isReadOnlyAttribute(String key)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, DoesNotExist, Timeout, NoSuccess;


    /**
     * Checks the attribute for being writable.
     * param key the attribute key.
     * return <code>true</code> if the attribute exists and is writable.
     */
    public boolean isWritableAttribute(String key)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, DoesNotExist, Timeout, NoSuccess;


    /**
     * Checks the attribute for being removable.
     * param key the attribute key.
     * return <code>true</code> if the attribute exists and is removable.
     */
    public boolean isRemovableAttribute(String key)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, DoesNotExist, Timeout, NoSuccess;


    /**
     * Checks the attribute for being a vector.
     * param key the attribute key.
     * return <code>true</code> if the attribute is a vector attribute.
     */
    public boolean isVectorAttribute(String key)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, DoesNotExist, Timeout, NoSuccess;
}
```

3.8.2 AsyncAttributes

```
package org.ogf.saga.attributes;

import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;

/**
 * Task versions of all methods from the <code>Attributes</code> interface.
 */
public interface AsyncAttributes extends Attributes {

    /**
     * Creates a task that sets an attribute to a value.
     * param mode determines the initial state of the task.
     * param key the attribute key.
     * param value value to set the attribute to.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task setAttribute(TaskMode mode, String key,
            String value) throws NotImplemented;

    /**
     * Creates a task that obtains the value of an attribute.
     * param mode determines the initial state of the task.
     * param key the attribute key.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task<String> getAttribute(TaskMode mode, String key)
        throws NotImplemented;

    /**
     * Creates a task that sets an attribute to an array of values.
     * param mode determines the initial state of the task.
     * param key the attribute key.
     * param values values to set the attribute to.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task setVectorAttribute(TaskMode mode, String key,
            String[] values)
        throws NotImplemented;
```

```
/**
 * Creates a task that obtains the array of values associated with an
 * attribute.
 * param mode determines the initial state of the task.
 * param key the attribute key.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<String[]> getVectorAttribute(TaskMode mode, String key)
    throws NotImplemented;


/**
 * Creates a task that removes an attribute.
 * param mode determines the initial state of the task.
 * param key the attribute key.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task removeAttribute(TaskMode mode, String key)
    throws NotImplemented;


/**
 * Creates a task that obtains the list of attribute keys.
 * param mode determines the initial state of the task.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<String[]> listAttributes(TaskMode mode)
    throws NotImplemented;


/**
 * Creates a task that finds matching attributes.
 * param mode determines the initial state of the task.
 * param patterns the search patterns.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<String[]> findAttributes(TaskMode mode, String... patterns)
    throws NotImplemented;


/**
 * Creates a task that checks the attribute mode for being read-only.
 * param mode determines the initial state of the task.
 * param key the attribute key.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
```

```
 *      method is not implemented.
 */
public Task<Boolean> isReadOnlyAttribute(TaskMode mode, String key)
    throws NotImplemented;

/**
 * Creates a task that checks the attribute mode for being writable.
 * param mode determines the initial state of the task.
 * param key the attribute key.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task<Boolean> isWritableAttribute(TaskMode mode, String key)
    throws NotImplemented;

/**
 * Creates a task that checks the attribute mode for being removable.
 * param mode determines the initial state of the task.
 * param key the attribute key.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task<Boolean> isRemovableAttribute(TaskMode mode, String key)
    throws NotImplemented;

/**
 * Creates a task that checks the attribute mode for being a vector.
 * param mode determines the initial state of the task.
 * param key the attribute key.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task<Boolean> isVectorAttribute(TaskMode mode, String key)
    throws NotImplemented;
}
```

3.9   SAGA Monitoring Model

The ability to query grid entities about state is requested in several SAGA use cases. Also, the SAGA task model introduces numerous new use cases for state monitoring.

This package definition approaches the problem space of monitoring to unify the various usage patterns (see details and examples), and to transparently incorporate SAGA task monitoring. The paradigm is realised by introducing monitorable SAGA objects, which expose *metrics* to the application, representing values to be monitored. Metrics thus represent monitorable entities.

A closely related topic is Computational Steering, which is (for our purposes) not seen independently from Monitoring: in the SAGA approach, the steering mechanisms extend the monitoring mechanisms with the ability to push values back to the monitored entity, i.e. to introduce writable metrics (see `fire()`). Thus, metrics can also represent steerable entities.

3.9.1   Metric

```
package org.ogf.saga.monitoring;

import org.ogf.saga.SagaObject;
import org.ogf.saga.attributes.Attributes;
import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.IncorrectState;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;

/**
 * Metrics represent monitorable entities.
 */
public interface Metric extends SagaObject, Attributes {

    /** Attribute name: name of the metric (ReadOnly). */
    public static final String NAME = "Name";

    /** Attribute name: description of the metric (ReadOnly). */
    public static final String DESCRIPTION = "Description";

    /**
     * Attribute name: access mode of the metric (ReadOnly).
```

```
    * Possible values: "ReadOnly", "ReadWrite", or "Final".
    * This determines what can be done with the VALUE attribute.
    */
   public static final String MODE = "Mode";

   /** Attribute name: unit of the metric (ReadOnly). */
   public static final String UNIT = "Unit";

   /**
    * Attribute name: value type of the metric (ReadOnly).
    * Possible values: "String", "Int", "Enum", "Float", "Bool",
    * "Time", "Trigger".
    */
   public static final String TYPE = "Type";

   /** Attribute name: value of the metric (See {link #MODE}). */
   public static final String VALUE = "Value";

   /**
    * Adds the specified callback to the metric.
    * return the cookie that identifies the callback in the metric.
    */
   public int addCallback(Callback cb)
       throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
           PermissionDenied, IncorrectState, Timeout, NoSuccess;

   /**
    * Removes a callback from the metric.
    * param cookie the cookie that identifies the metric.
    */
   public void removeCallback(int cookie)
       throws NotImplemented, BadParameter, AuthenticationFailed,
           AuthorizationFailed, PermissionDenied, Timeout,
           NoSuccess;

   /**
    * Pushes the metric value to the backend.
    */
   public void fire()
       throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
           PermissionDenied, IncorrectState, Timeout, NoSuccess;
}
```

3.9.2   Callback

```
package org.ogf.saga.monitoring;

import org.ogf.saga.context.Context;
```

```
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.NotImplemented;

/**
 * Asynchronous handler for metric changes.
 */
public interface Callback {
    /**
     * Asynchronous handler for metric changes.
     * param mt the SAGA Monitorable object which causes the callback
     *     invocation.
     * param metric the metric causing the callback invocation.
     * param ctx the context associated with the callback causing entity.
     * return wether the callback stays registered.
     */
    public boolean cb(Monitorable mt, Metric metric, Context ctx)
        throws NotImplemented, AuthorizationFailed;
}
```

3.9.3   Monitorable

```
package org.ogf.saga.monitoring;

import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.DoesNotExist;
import org.ogf.saga.error.IncorrectState;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;

/**
 * The <code>Monitorable</code> interface is implemented by SAGA objects
 * that can be monitored (have one or more associated metrics).
 */
public interface Monitorable {

    /**
     * Lists all metrics associated with the object.
     * return the names identifying the metrics associated with the object.
     */
    public String[] listMetrics()
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, Timeout, NoSuccess;

    /**
```

```
    * Returns a metric instance, identified by name.
    * param name the name of the metric to be returned.
    * return the metric.
    */
   public Metric getMetric(String name)
       throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
           PermissionDenied, DoesNotExist, Timeout, NoSuccess;

   /**
    * Adds a callback to the specified metric.
    * param name identifier the metric to which the callback is to be added.
    * param cb the callback to be added.
    * return a handle to be used for removal of the callback.
    */
   public int addCallback(String name, Callback cb)
       throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
           PermissionDenied, DoesNotExist, Timeout, NoSuccess, IncorrectState;

   /**
    * Removes the specified callback.
    * param name identifier the metric from which the callback is to be
    *     removed.
    * param cookie identifies the callback to be removed.
    */
   public void removeCallback(String name, int cookie)
       throws NotImplemented, DoesNotExist, BadParameter, Timeout, NoSuccess,
           AuthenticationFailed, AuthorizationFailed, PermissionDenied;
}
```

3.9.4   AsyncMonitorable

```
package org.ogf.saga.monitoring;

import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;

/**
 * This interface defines the task versions of the <code>Monitorable</code>
 * interface. Needed for streams.
 */
public interface AsyncMonitorable extends Monitorable {

   /**
    * Creates a task that lists all metrics associated with the object.
    * param mode the task mode.
    * return the task.
    * exception NotImplemented is thrown when the task version of this
```

```
 *      method is not implemented.
 */
public Task<String[]> listMetrics(TaskMode mode)
    throws NotImplemented;

/**
 * Creates a task that obtains a metric instance, identified by name.
 * param mode the task mode.
 * param name the name of the metric to be returned.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task<Metric> getMetric(TaskMode mode, String name)
    throws NotImplemented;

/**
 * Creates a task that adds a callback to the specified metric.
 * param mode the task mode.
 * param name identifier the metric to which the callback is to be added.
 * param cb the callback to be added.
 * return a handle to be used for removal of the callback.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task<Integer> addCallback(TaskMode mode, String name, Callback cb)
    throws NotImplemented;

/**
 * Creates a task that removes the specified callback.
 * param mode the task mode.
 * param name identifier the metric from which the callback is to be
 *      removed.
 * param cookie identifies the callback to be removed.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task removeCallback(TaskMode mode, String name, int cookie)
    throws NotImplemented;
}
```

3.9.5   Steerable

```
package org.ogf.saga.monitoring;

import org.ogf.saga.error.AlreadyExists;
import org.ogf.saga.error.AuthenticationFailed;
```

```
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.DoesNotExist;
import org.ogf.saga.error.IncorrectState;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;

/**
 * The <code>Steerable</code> interface is implemented by SAGA objects
 * that can be steered (have writable metrics and/or allow to add new metrics).
 */
public interface Steerable {
    /**
     * Adds a metric instance to the application instance.
     * param metric the metric instance to be added.
     * return success or failure.
     */
    public boolean addMetric(Metric metric)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, AlreadyExists, Timeout, NoSuccess;

    /**
     * Removes a metric instance.
     * param name the name of the metric to be removed.
     */
    public void removeMetric(String name)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, DoesNotExist, Timeout, NoSuccess;

    /**
     * Pushes a new metric value to the backend.
     * param name the name of the metric to be fired.
     */
    public void fireMetric(String name)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, IncorrectState, DoesNotExist, Timeout,
            NoSuccess;
}
```

3.9.6   AsyncSteerable

```
package org.ogf.saga.monitoring;

import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;
```

```
/**
 * This interface specifies the Async versions of the methods in
 * <code>Steerable</code>. Needed for job.JobSelf.
 */
public interface AsyncSteerable extends Steerable {
    /**
     * Creates a task that adds a metric instance to the application instance.
     * param mode the task mode.
     * param metric the metric instance to be added.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task<Boolean> addMetric(TaskMode mode, Metric metric)
        throws NotImplemented;

    /**
     * Creates a task that removes a metric instance.
     * param mode the task mode.
     * param name the name of the metric to be removed.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task removeMetric(TaskMode mode, String name)
        throws NotImplemented;

    /**
     * Creates a task that pushes a new metric value to the backend.
     * param mode the task mode.
     * param name the name of the metric to be fired.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task fireMetric(TaskMode mode, String name)
        throws NotImplemented;
}
```

3.9.7   MonitoringFactory

```
package org.ogf.saga.monitoring;

import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.Timeout;
```

```
/**
 * Factory for objects in the monitoring package.
 */
public abstract class MonitoringFactory {

    private static MonitoringFactory factory;

    private synchronized static void initializeFactory()
        throws NotImplemented {
        if (factory == null) {
            factory = ImplementationBootstrapLoader.createMonitoringFactory();
        }
    }

    /**
     * Constructs a <code>Metric</code> object with the specified parameters.
     * This method is to be provided by the factory.
     * param name name of the metric.
     * param desc description of the metric.
     * param mode mode of the metric.
     * param unit unit of the metric value.
     * param type type of the metric.
     * param value value of the metric.
     * return the metric.
     */
    protected abstract Metric doCreateMetric(String name, String desc,
            String mode, String unit, String type, String value)
        throws NotImplemented, BadParameter, Timeout, NoSuccess;

    /**
     * Constructs a <code>Metric</code> object with the specified parameters.
     * param name name of the metric.
     * param desc description of the metric.
     * param mode mode of the metric.
     * param unit unit of the metric value.
     * param type type of the metric.
     * param value value of the metric.
     * return the metric.
     */
    public synchronized static Metric createMetric(String name, String desc,
            String mode, String unit, String type, String value)
        throws NotImplemented, BadParameter, Timeout, NoSuccess {
        initializeFactory();
        return factory.doCreateMetric(name, desc, mode, unit, type, value);
    }
}
```

3.10 SAGA Task Model

Operations performed in highly heterogenous distributed environments may take a long time to complete, and it is thus desirable to have the ability to perform operations in an asynchronous manner. The SAGA task model as described here, provides this ability to all other SAGA classes. As such, the package is orthogonal to the rest of the SAGA API.

For a detailed description of the SAGA task model, we refer to the GFD.90 document. Here, we only discuss Java-specific issues.

Tasks versus Threads   Tasks and (Java) threads are having some potential for confusion. In SAGA, tasks have a semantically richer meaning. In particular, threads always imply that the state management for the asynchronous operation lies within the application hosting the thread. SAGA tasks, however, imply no such restriction. This does not mean that threads cannot be used to implement tasks, but it does mean that that may not always be the optimal solution.

In the `Task` interface, as well as in the `TaskContainer` interface, the `wait` method is renamed to `waitFor`, because it is not supposed to redefine the `java.lang.Object wait` method.

Java has the `java.util.concurrent` package, which contains the `Future` interface, which has methods that are similar to some of the methods in `Task`. Therefore, it was decided that the `Task` interface extends `Future`, which makes all methods from the `Future` interface available, in addition to the ones specified in the `Task` interface itself. This increases the burden on the implementors a bit, but not too much, because of the similarity.

Tasks and Error Handling   Errors arising from synchronous method invocations on SAGA objects are, in general, flagged by exceptions. For operations invoked through a task, any exception thrown by the operation is stored in the task, which then changes its state to `Failed`. The `rethrow` method makes the exception available to the application.

3.10.1 Async

```
package org.ogf.saga.task;

/**
 * This interface is empty on purpose, and is used only
 * for tagging of SAGA classes which implement the SAGA
 * task model.
```

```
 */
public interface Async {
    // nothing here.
}
```

3.10.2   Task

```
package org.ogf.saga.task;

import java.util.concurrent.Future;

import org.ogf.saga.SagaObject;
import org.ogf.saga.error.AlreadyExists;
import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.DoesNotExist;
import org.ogf.saga.error.IncorrectState;
import org.ogf.saga.error.IncorrectURL;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;
import org.ogf.saga.monitoring.Monitorable;

/**
 * Tasks can only be created through asynchronous method calls.
 * The generic parameter <code>E</code> denotes the type of the return
 * value of the asynchronous method call.
 */
public interface Task<E> extends SagaObject, Monitorable, Future<E> {

    /**
     * Metric name: fires on task state change, and has the literal value
     * of the task state enumeration.
     */
    public static final String TASK_STATE = "Task.state";

    /**
     * Starts the asynchronous operation.
     */
    public void run()
        throws NotImplemented, IncorrectState, Timeout, NoSuccess;

    /**
     * Waits for the task end up in a final state.
     * The SAGA API specifies that this method is called <code>wait</code>,
     * for Java the name needs to be changed slightly.
```

```
 */
public void waitFor()
    throws NotImplemented, IncorrectState, Timeout, NoSuccess;

/**
 * Waits for the task to end up in a final state.
 * The SAGA API specifies that this method is called <code>wait</code>,
 * for Java the name needs to be changed slightly.
 * param timeoutInSeconds maximum number of seconds to wait.
 * return <code>true</code> if the task is finished within the specified time.
 */
public boolean waitFor(float timeoutInSeconds)
    throws NotImplemented, IncorrectState, Timeout, NoSuccess;

/**
 * Cancels the asynchronous operation.
 * This is a non-blocking version, which may continue to try and cancel
 * the task in the background. The task state will remain RUNNING until the
 * cancel operation succeeds.
 */
public void cancel()
    throws NotImplemented, IncorrectState, Timeout, NoSuccess;

/**
 * Cancels the asynchronous operation.
 * If it does not succeed to cancel the task within the specified timeout,
 * it may continue to try and cancel the task in the background.
 * The task state will remain RUNNING until the cancel operation succeeds.
 * param timeoutInSeconds maximum time for freeing resources.
 */
public void cancel(float timeoutInSeconds)
    throws NotImplemented, IncorrectState, Timeout, NoSuccess;

/**
 * Gets the state of the task.
 * return the state of the task.
 */
public State getState()
    throws NotImplemented, Timeout, NoSuccess;

/**
 * Obtains the result of the asynchronous method call.
 * return the result.
 * exception IncorrectState is thrown when the task is not in state DONE.
 */
public E getResult() throws NotImplemented, IncorrectState, Timeout,
        NoSuccess;

/**
 * Gets the object from which the task was created.
```

```
     * return the object this task was created from.
     */
    public <T> T getObject()
        throws NotImplemented, Timeout, NoSuccess;


    /**
     * Throws any exception a failed task caught.
     */
    public void rethrow()
        throws NotImplemented, IncorrectURL,
            AuthenticationFailed, AuthorizationFailed, PermissionDenied,
            BadParameter, IncorrectState, AlreadyExists, DoesNotExist,
            Timeout, NoSuccess;
}
```

3.10.3   TaskContainer

```
package org.ogf.saga.task;

import org.ogf.saga.SagaObject;
import org.ogf.saga.error.DoesNotExist;
import org.ogf.saga.error.IncorrectState;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.Timeout;
import org.ogf.saga.monitoring.Monitorable;

/**
 * Container object for tasks.
 */
public interface TaskContainer extends SagaObject, Monitorable {

    /**
     * Metric name: fires on state changes of any task in the container,
     * and has the value of that task's handle.
     */
    public static final String TASKCONTAINER_STATE = "TaskContainer.state";

    /**
     * Adds a task to the task container.
     * param task the task to add.
     * return a handle allowing for removal of the task.
     */
    public int add(Task task) throws NotImplemented, Timeout, NoSuccess;

    /**
     * Removes the task identified by the specified cookie from this container.
     * param cookie identifies the task.
```

```
 * return the task.
 */
public <T> Task<T> remove(int cookie)
    throws NotImplemented, DoesNotExist, Timeout, NoSuccess;


/**
 * Starts all asynchronous operations in the container.
 */
public void run()
    throws NotImplemented, IncorrectState, DoesNotExist, Timeout, NoSuccess;


/**
 * Waits for one or more of the tasks to end up in a final state.
 * This method blocks indefinately. One of the finished tasks is returned,
 * and removed from the task container.
 * param mode wait for ALL or ANY task.
 * return any of the finished tasks.
 */
public Task waitFor(WaitMode mode)
    throws NotImplemented, IncorrectState, DoesNotExist, Timeout, NoSuccess;


/**
 * Waits for one or more of the tasks to end up in a final state.
 * One of the finished tasks is returned, and removed from the task
 * container. If none of the tasks is finished within the specified
 * timeout, <code>null</code> is returned.
 * param timeoutInSeconds number of seconds to wait.
 * param mode wait for ALL or ANY task.
 * return the finished tasks.
 */
public Task waitFor(float timeoutInSeconds, WaitMode mode)
    throws NotImplemented, IncorrectState, DoesNotExist, Timeout, NoSuccess;


/**
 * Cancels all the asynchronous operations in the container.
 * This is a non-blocking version, which may continue to try and cancel
 * tasks in the background. The task states will remain RUNNING until the
 * cancel operation succeeds.
 */
public void cancel()
    throws NotImplemented, IncorrectState, DoesNotExist, Timeout, NoSuccess;


/**
 * Cancels all the asynchronous operations in the container.
 * When the timeout expires, this version may continue to try and cancel
 * tasks in the background. The task states will remain RUNNING until the
 * cancel operation succeeds.
 * param timeoutInSeconds time for freeing resources.
 */
public void cancel(float timeoutInSeconds)
```

```
        throws NotImplemented, IncorrectState, DoesNotExist, Timeout, NoSuccess;

    /**
     * Returns the number of tasks in this task container.
     * return the number of tasks.
     */
    public int size()
        throws NotImplemented, Timeout, NoSuccess;

    /**
     * Lists the tasks in this task container.
     * return an array of cookies.
     */
    public int[] listTasks()
        throws NotImplemented, Timeout, NoSuccess;

    /**
     * Gets a single task from the task container.
     * param cookie the integer identifying the task.
     * return the task.
     */
    public <T> Task<T> getTask(int cookie)
        throws NotImplemented, DoesNotExist, Timeout, NoSuccess;

    /**
     * Gets the tasks in this task container.
     * return the tasks.
     */
    public Task[] getTasks()
        throws NotImplemented, Timeout, NoSuccess;

    /**
     * Gets the states of all tasks in the task container.
     * return the states.
     */
    public State[] getStates()
        throws NotImplemented, Timeout, NoSuccess;
}
```

3.10.4 Taskfactory

```
package org.ogf.saga.task;

import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.Timeout;
```

```
/**
 * Factory for objects in the task package.
 */
public abstract class TaskFactory {

    private static TaskFactory factory;

    private synchronized static void initializeFactory()
        throws NotImplemented {
        if (factory == null) {
            factory = ImplementationBootstrapLoader.createTaskFactory();
        }
    }

    /**
     * Constructs a <code>TaskContainer</code> object.
     * This method is to be provided by the factory.
     * return the task container.
     */
    protected abstract TaskContainer doCreateTaskContainer()
        throws NotImplemented, Timeout, NoSuccess;

    /**
     * Constructs a <code>TaskContainer</code> object.
     * This method is to be provided by the factory.
     * return the task container.
     */
    public synchronized static TaskContainer createTaskContainer()
        throws NotImplemented, Timeout, NoSuccess {
        initializeFactory();
        return factory.doCreateTaskContainer();
    }
}
```

3.10.5   State

```
package org.ogf.saga.task;

/**
 * This enumeration type describes the possible states of a task.
 * Also includes the states of the job class, since enumerations cannot
 * be extended.
 */
public enum State {

    NEW (1),
    RUNNING (2),
    DONE (3),
```

```
    CANCELED (4),
    FAILED (5),
    SUSPENDED(6);

    private int value;

    State (int value) {
        this.value = value;
    }

    /**
     * Returns the integer value of this enumeration literal.
     * return the value.
     */
    public int getValue() {
        return value;
    }
}
```

3.10.6   TaskMode

```
package org.ogf.saga.task;

/**
 * This enumeration type describes the possible ways to create a task:
 * Asynchronous (the task is started in RUNNING state), Task (the task is
 * started in NEW state), or Synchronous (the task is started and waited for).
 */
public enum TaskMode {

    ASYNC (1),
    TASK (2),
    SYNC (3);

    private int value;

    TaskMode (int value) {
        this.value = value;
    }

    /**
     * Returns the integer value of this enumeration literal.
     * return the value.
     */
    public int getValue() {
        return value;
    }
}
```

3.10.7   WaitMode

```
package org.ogf.saga.task;

/**
 * When waiting for tasks in a task container, the user can either wait
 * for all tasks in the container, or for any task in the container.
 */
public enum WaitMode {
    ALL (0),
    ANY (1);

    private int value;

    WaitMode(int value) {
        this.value = value;
    }

    /**
     * Returns the integer value of this enumeration literal.
     * return the value.
     */
    public int getValue() {
        return value;
    }
}
```

4   SAGA API Specification – API Packages

The Functional SAGA API packages define the functional SAGA API scope, as motivated in GFD.90 [3] and in GFD.71 [7].

The interfaces, classes and methods defined in this part of the specification are, in general, representing explicit entities and actions of some backend system. As such, all operations on these entities are, in general, subject to authentication and authorization. In order to simplify the specification, the following exceptions are not separately motivated: `AuthenticationFailed`, `Authorization-Failed`, `PermissionDenied`, `Timeout`, `NoSuccess`. These exceptions have then exactly the semantics as indicated in their description in Section 3.1. Additionally, the conventions for the exceptions `NotImplemented` and `IncorrectURL` apply as described in Section 3.

4.1   SAGA Job Management

This section describes the SAGA API for submitting jobs to a grid resource, either in batch mode, or in an interactive mode. It also describes how to control these submitted jobs (e.g. to `cancel()`, `suspend()`, or `signal()` a running job), and how to retrieve status information for both running and completed jobs.

The API covers four interfaces: `job_description`, `job_service`, `job` and `job_self`. The job description class is a container for a well defined set of attributes which, using JSDL [5] based keys, defines the job to be started, and its runtime and resource requirements. The job server represents a resource management endpoint which allows the starting and inspection of jobs.

The `job` interface itself is central to the API, and represents an application instance running under the management of a resource manager. The `job_self` interface `IS-A` job, but additionally implements the steering interface. The purpose of this interface is to represent the current SAGA application, which allows for a number of use cases with applications which actively interact with the grid infrastructure, for example to provide steering capabilities, to migrate itself, or to set new job attributes.

The job interface inherits the `task` interface 3.10, and uses its methods to `run()`, `wait()` for, and to `cancel()` jobs. The inheritance feature also allows for the management of large numbers of jobs in task containers. Additional methods provided by the `job` interface relate to the `Suspended` state (which is not available on tasks), and provide access to the job's standard I/O streams, and to more detailed status information.

4.1.1   Job

```
package org.ogf.saga.job;

import java.io.InputStream;
import java.io.OutputStream;

import org.ogf.saga.attributes.AsyncAttributes;
import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.DoesNotExist;
import org.ogf.saga.error.IncorrectState;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;
import org.ogf.saga.permissions.Permissions;
import org.ogf.saga.task.Async;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;

public interface Job extends Task<Object>, Async, AsyncAttributes, Permissions {

    // Required attributes:

    /** Attribute name, SAGA representation of the job identifier. */
    public static final String JOBID = "JobID";

    // Optional attributes:

    /**
     * Attribute name, list of host names or IP addresses allocated to run
     * this job.
     */
    public static final String EXECUTIONHOSTS = "ExecutionHosts";

    /**
     * Attribute name, time stamp of the job creation in the resource manager.
     */
    public static final String CREATED = "Created";

    /** Attribute name, time stamp indicating when the job started running. */
    public static final String STARTED = "Started";

    /** Attribute name, time stamp indicating when the job finished. */
    public static final String FINISHED = "Finished";

    /** Attribute name, working directory on the execution host. */
```

```
public static final String WORKINGDIRECTORY = "WorkingDirectory";

/** Attribute name, process exit code. */
public static final String EXITCODE = "ExitCode";

/** Attribute name, signal number which caused the job to exit. */
public static final String TERMSIG = "Termsig";

// Required metrics:

/** Metric name, fires on state changes of the job. */
public static final String JOB_STATE = "job.state";

// Optional metrics:

/** Metric name, fires as a job changes its state detail. */
public static final String JOB_STATEDETAIL = "job.state_detail";

/** Metric name, fires as a job receives a signal. */
public static final String JOB_SIGNAL = "job.signal";

/** Metric name, number of CPU seconds consumed by the job. */
public static final String JOB_CPUTIME = "job.cpu_time";

/** Metric name, current aggregate memory usage of the job. */
public static final String JOB_MEMORYUSE = "job.memory_use";

/** Metric name, current aggregate virtual memory usage of the job. */
public static final String JOB_VMEMORYUSE = "job.vmemory_use";

/** Metric name, current performance. */
public static final String JOB_PERFORMANCE = "job.performance";

// Methods

/**
 * Retrieves the job description that was used to submit this job
 * instance.
 * return the job description
 */
public JobDescription getJobDescription()
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, DoesNotExist, Timeout, NoSuccess;

/**
 * Returns the input stream of this job (to which can be written).
 * return the stream.
 */
public OutputStream getStdin()
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
```

```
            PermissionDenied, BadParameter, DoesNotExist, Timeout,
            IncorrectState, NoSuccess;

    /**
     * Returns the output stream of this job (which can be read).
     * return the stream.
     */
    public InputStream getStdout()
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, BadParameter, DoesNotExist, Timeout,
            IncorrectState, NoSuccess;

    /**
     * Returns the error stream of this job (which can be read).
     * return the stream.
     */
    public InputStream getStderr()
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, BadParameter, DoesNotExist, Timeout,
            IncorrectState, NoSuccess;

    /**
     * Asks the resource manager to perform a suspend operation on a
     * running job.
     */
    public void suspend()
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, IncorrectState, Timeout, NoSuccess;

    /**
     * Asks the resource manager to perform a resume operation on a
     * suspended job.
     */
    public void resume()
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, IncorrectState, Timeout, NoSuccess;

    /**
     * Asks the resource manager to initiate a checkpoint operation on a
     * running job.
     */
    public void checkpoint()
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, IncorrectState, Timeout, NoSuccess;

    /**
     * Asks the resource manager to migrate a job.
     * param jd new job parameters to apply when the job is migrated.
     */
    public void migrate(JobDescription jd)
```

```
      throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
          PermissionDenied, BadParameter, IncorrectState, Timeout, NoSuccess;

  /**
   * Asks the resource manager to deliver an arbitrary signal to a
   * dispatched job.
   */
  public void signal(int signum)
      throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
          PermissionDenied, BadParameter, IncorrectState, Timeout, NoSuccess;

  //
  // Task versions ...
  //

  /**
   * Creates a task that retrieves the job description that was used to
   * submit this job instance.
   * param mode the task mode.
   * return the task.
   * exception NotImplemented is thrown when the task version of this
   *     method is not implemented.
   */
  public Task<JobDescription> getJobDescription(TaskMode mode)
      throws NotImplemented;

  /**
   * Creates a task that obtains the input stream of this job (to which can
   * be written).
   * param mode the task mode.
   * return the task.
   * exception NotImplemented is thrown when the task version of this
   *     method is not implemented.
   */
  public Task<OutputStream> getStdin(TaskMode mode)
      throws NotImplemented;

  /**
   * Creates a task that obtains the output stream of this job (which can
   * be read).
   * param mode the task mode.
   * return the task.
   * exception NotImplemented is thrown when the task version of this
   *     method is not implemented.
   */
  public Task<InputStream> getStdout(TaskMode mode)
      throws NotImplemented;

  /**
   * Creates a task that obtains the error stream of this job (which can
```

```
 * be read).
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<InputStream> getStderr(TaskMode mode)
    throws NotImplemented;


/**
 * Creates a task that asks the resource manager to perform a suspend
 * operation on a running job.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task suspend(TaskMode mode)
    throws NotImplemented;


/**
 * Creates a task that asks the resource manager to perform a resume
 * operation on a suspended job.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task resume(TaskMode mode)
    throws NotImplemented;


/**
 * Creates a task that asks the resource manager to initiate a checkpoint
 * operation on a running job.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task checkpoint(TaskMode mode)
    throws NotImplemented;


/**
 * Creates a task that asks the resource manager to migrate a job.
 * param jd new job parameters to apply when the job is migrated.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
```

```
    public Task migrate(TaskMode mode, JobDescription jd)
        throws NotImplemented;


    /**
     * Creates a task that asks the resource manager to deliver an arbitrary
     * signal to a dispatched job.
     * param mode the task mode.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task signal(TaskMode mode, int signum)
        throws NotImplemented;
}
```

4.1.2   JobDescription

```
package org.ogf.saga.job;

import org.ogf.saga.SagaObject;
import org.ogf.saga.attributes.Attributes;


/**
 * Contents of a job description is defined by its attributes.
 * Should we have separate methods for each of the attributes???
 */
public interface JobDescription extends SagaObject, Attributes {

    // Required attributes:

    /** Attribute name, command to execute. */
    public static final String EXECUTABLE = "Executable";

    // Optional attributes:

    /** Attribute name, positional parameters for the command. */
    public static final String ARGUMENTS = "Arguments";

    /** Attribute name, SPMD job type and startup mechanism. */
    public static final String SPMDVARIATION = "SPMDVariation";

    /** Attribute name, total number of cpus requested for this job. */
    public static final String TOTALCPUCOUNT = "TotalCPUCount";

    /** Attribute name, total number of processes to be started. */
    public static final String NUMBEROFPROCESSES = "NumberOfProcesses";

    /** Attribute name, total number of processes to be started per host. */
```

```
public static final String PROCESSESPERHOST = "ProcessesPerHost";

/** Attribute name, number of threads to start per process. */
public static final String THREADSPERPROCESS = "ThreadsPerProcess";

/** Attribute name, set of environment variables for the job. */
public static final String ENVIRONMENT = "Environment";

/** Attribute name, working directory for the job. */
public static final String WORKINGDIRECTORY = "WorkingDirectory";

/** Attribute name, run the job in interactive mode. */
public static final String INTERACTIVE = "Interactive";

/** Attribute name, pathname of the standard input file. */
public static final String INPUT = "Input";

/** Attribute name, pathname of the standard output file. */
public static final String OUTPUT = "Output";

/** Attribute name, pathname of the standard error file. */
public static final String ERROR = "Error";

/** Attribute name, a list of file transfer directives. */
public static final String FILETRANSFER = "FileTransfer";

/**
 * Attribute name, defines whether output files get removed after the
 * job finishes.
 */
public static final String CLEANUP = "Cleanup";

/** Attribute name, time at which the job should be scheduled. */
public static final String JOBSTARTTIME = "JobStartTime";

/**
 * Attribute name, estimate of total number of CPU seconds the job will
 * require.
 */
public static final String TOTALCPUTIME = "TotalCPUTime";

/** Attribute name, estimate of the amount of memory the job requires. */
public static final String TOTALPHYSICALMEMORY = "TotalPhysicalMemory";

/** Attribute name, compatible processor for job submission. */
public static final String CPUARCHITECTURE = "CPUArchitecture";

/** Attribute name, compatible operating system for job submission. */
public static final String OPERATINGSYSTEMTYPE = "OperatingSystemType";
```

```
    /**
     * Attribute name, list of host names which are to be considered by the
     * resource manager as candidate targets.
     */
    public static final String CANDIDATEHOSTS = "CandidateHosts";

    /** Attribute name, name of queue to place the job into. */
    public static final String QUEUE = "Queue";

    /**
     * Attribute name, set of end points where to report job state transitions.
     */
    public static final String JOBCONTACT = "JobContact";
}
```

4.1.3   JobSelf

```
package org.ogf.saga.job;

import org.ogf.saga.monitoring.AsyncSteerable;

/**
 * A JobSelf is a Job that represents the current application, and is
 * steerable.
 */
public interface JobSelf extends Job, AsyncSteerable {
}
```

4.1.4   Jobservice

```
package org.ogf.saga.job;

import java.util.List;

import org.ogf.saga.SagaObject;
import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.DoesNotExist;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;
import org.ogf.saga.task.Async;
import org.ogf.saga.task.TaskMode;
import org.ogf.saga.task.Task;
```

```
/**
 * A JobService represents a resource management back-end.
 *
 * It allows for job creation, submission, and discovery.
 * Deviation from the SAGA specification: the convenience method
 * <code>runJob</code> is specified differently here, because
 * as described in the SAGA specifications it cannot
 * easily be specified in Java, since Java has no OUT parameters.
 */
public interface JobService extends SagaObject, Async {

    /**
     * Creates a job instance as specified by the job description provided.
     * The job is delivered in 'New' state. The provided job description
     * is copied, so can be modified after this call.
     * param jd the job description.
     * return the job.
     */
    public Job createJob(JobDescription jd) throws NotImplemented,
            AuthenticationFailed, AuthorizationFailed, PermissionDenied,
            BadParameter, Timeout, NoSuccess;

    /**
     * Runs the specified command on the specified host.
     * Deviation from the SAGA specification: the input, output and
     * error stream OUT parameters are not specified here, since Java
     * has no OUT parameters. Unfortunately, their absence, according
     * to the SAGA specifications, implies a non-interactive job.
     * Since interactive jobs should still be supported, a parameter
     * is added here to specify whether the job is interactive.
     * If interactive, the streams can be obtained from the Job
     * using the {link Job#getStdin()}, {@link Job#getStdout()},
     * and {link Job#getStderr()} methods.
     * param commandLine the command to run.
     * param host hostname of the host on which the command must be run.
     *     If this is an empty string, the implementation is free to choose
     *     a host.
     * param interactive specifies whether the job is interactive.
     * return the job.
     */
    public Job runJob(String commandLine, String host, boolean interactive)
            throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, BadParameter, Timeout, NoSuccess;

    /**
     * Runs the specified command, non-interactively, on the specified host.
     * Deviation from the SAGA specification: the input, output and
     * error stream OUT parameters are not specified here, since Java
     * has no OUT parameters.
```

```
 * param commandLine the command to run.
 * param host hostname of the host on which the command must be run.
 *     If this is an empty string, the implementation is free to choose
 *     a host.
 * return the job.
 */
public Job runJob(String commandLine, String host)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, Timeout, NoSuccess;

/**
 * Runs the specified command on a host chosen by the implementation.
 * Deviation from the SAGA specification: the input, output and
 * error stream OUT parameters are not specified here, since Java
 * has no OUT parameters. Unfortunately, their absence, according
 * to the SAGA specifications, implies a non-interactive job.
 * Since interactive jobs should still be supported, a parameter
 * is added here to specify whether the job is interactive.
 * If interactive, the streams can be obtained from the Job
 * using the {link Job#getStdin()}, {@link Job#getStdout()},
 * and {link Job#getStderr()} methods.
 * param commandLine the command to run.
 * param interactive specifies whether the job is interactive.
 * return the job.
 */
public Job runJob(String commandLine, boolean interactive)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, Timeout, NoSuccess;

/**
 * Runs the specified command, non-interactively,
 * on a host chosen by the implementation.
 * Deviation from the SAGA specification: the input, output and
 * error stream OUT parameters are not specified here, since Java
 * has no OUT parameters.
 * param commandLine the command to run.
 * return the job.
 */
public Job runJob(String commandLine)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, Timeout, NoSuccess;

/**
 * Obtains the list of jobs that are currently known to the
 * resource manager.
 * return a list of job identifications.
 */
public List<String> list() throws NotImplemented, AuthenticationFailed,
        AuthorizationFailed, PermissionDenied, Timeout, NoSuccess;
```

```
/**
 * Returns the job instance associated with the specified job
 * identification.
 * param jobId the job identification.
 * return the job instance.
 */
public Job getJob(String jobId) throws NotImplemented,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, DoesNotExist, Timeout, NoSuccess;


/**
 * Returns a job instance representing the calling application.
 * return the job instance.
 */
public JobSelf getSelf() throws NotImplemented,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        Timeout, NoSuccess;


/**
 * Creates a task that creates a job instance as specified by the
 * job description provided.
 * The job is delivered in 'New' state.
 * param mode the task mode.
 * param jd the job description.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<Job> createJob(TaskMode mode, JobDescription jd)
    throws NotImplemented;


/**
 * Creates a task that obtains the list of jobs that are currently known
 * to the resource manager.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<List<String>> list(TaskMode mode)
    throws NotImplemented;


/**
 * Creates a task that obtains the job instance associated with the
 * specified job identification.
 * param mode the task mode.
 * param jobId the job identification.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
```

```
     */
    public Task<Job> getJob(TaskMode mode, String jobId)
        throws NotImplemented;

    /**
     * Creates a task that obtains a job instance representing the calling
     * application.
     * param mode the task mode.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task<JobSelf> getSelf(TaskMode mode)
        throws NotImplemented;

    /**
     * Creates a task that runs the specified command on the specified host.
     * Deviation from the SAGA specification: the input, output and
     * error stream OUT parameters are not specified here, since Java
     * has no OUT parameters. Unfortunately, their absence, according
     * to the SAGA specifications, implies a non-interactive job.
     * Since interactive jobs should still be supported, a parameter
     * is added here to specify whether the job is interactive.
     * If interactive, the streams can be obtained from the Job
     * using the {link Job#getStdin()}, {@link Job#getStdout()},
     * and {link Job#getStderr()} methods
     * param mode the task mode.
     * param commandLine the command to run.
     * param host hostname of the host on which the command must be run.
     *     If this is an empty string, the implementation is free to choose
     *     a host.
     * param interactive specifies whether the job is interactive.
     * return the task.
     */
    public Task<Job> runJob(TaskMode mode, String commandLine, String host,
            boolean interactive) throws NotImplemented;

    /**
     * Creates a task that runs the specified command,
     * non-interactively, on the specified host.
     * Deviation from the SAGA specification: the input, output and
     * error stream OUT parameters are not specified here, since Java
     * has no OUT parameters.
     * param mode the task mode.
     * param commandLine the command to run.
     * param host hostname of the host on which the command must be run.
     *     If this is an empty string, the implementation is free to choose
     *     a host.
     * return the task.
     */
```

```
    public Task<Job> runJob(TaskMode mode, String commandLine, String host)
            throws NotImplemented;

    /**
     * Creates a task that runs the specified command on a host
     * chosen by the implementation.
     * Deviation from the SAGA specification: the input, output and
     * error stream OUT parameters are not specified here, since Java
     * has no OUT parameters. Unfortunately, their absence, according
     * to the SAGA specifications, implies a non-interactive job.
     * Since interactive jobs should still be supported, a parameter
     * is added here to specify whether the job is interactive.
     * If interactive, the streams can be obtained from the Job
     * using the {link Job#getStdin()}, {@link Job#getStdout()},
     * and {link Job#getStderr()} methods.
     * param mode the task mode.
     * param commandLine the command to run.
     * param interactive specifies whether the job is interactive.
     * return the task.
     */
    public Task<Job> runJob(TaskMode mode, String commandLine, boolean interactive)
            throws NotImplemented;

    /**
     * Creates a task that runs the specified command, non-interactively,
     * on a host chosen by the implementation.
     * Deviation from the SAGA specification: the input, output and
     * error stream OUT parameters are not specified here, since Java
     * has no OUT parameters.
     * param mode the task mode.
     * param commandLine the command to run.
     * return the task.
     */
    public Task<Job> runJob(TaskMode mode, String commandLine)
            throws NotImplemented;
}
```

4.1.5 JobFactory

```
package org.ogf.saga.job;

import org.ogf.saga.URL;
import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.IncorrectURL;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
```

```
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.SagaError;
import org.ogf.saga.error.Timeout;
import org.ogf.saga.session.Session;
import org.ogf.saga.session.SessionFactory;
import org.ogf.saga.task.TaskMode;
import org.ogf.saga.task.Task;

/**
 * Factory for objects from the job package.
 */
public abstract class JobFactory {

    private static JobFactory factory;

    private static synchronized void initializeFactory()
        throws NotImplemented {
        if (factory == null) {
            factory = ImplementationBootstrapLoader.createJobFactory();
        }
    }

    /**
     * Creates a job description. To be provided by the implementation.
     * return the job description.
     */
    protected abstract JobDescription doCreateJobDescription()
        throws NotImplemented, NoSuccess;

    /**
     * Creates a job service. To be provided by the implementation.
     * param session the session handle.
     * param rm contact string for the resource manager.
     * return the job service.
     */
    protected abstract JobService doCreateJobService(
            Session session, URL rm)
        throws NotImplemented, IncorrectURL,
            AuthenticationFailed, AuthorizationFailed, PermissionDenied,
            Timeout, NoSuccess;

    /**
     * Creates a task that creates a job service.
     * To be provided by the implementation.
     * param mode the task mode.
     * param session the session handle.
     * param rm contact string for the resource manager.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
```

```
 */
protected abstract Task<JobService> doCreateJobService(
        TaskMode mode, Session session, URL rm)
    throws NotImplemented;

/**
 * Creates a job description.
 * return the job description.
 */
public static JobDescription createJobDescription()
    throws NotImplemented, NoSuccess {
    initializeFactory();
    return factory.doCreateJobDescription();
}

/**
 * Creates a job service.
 * param session the session handle.
 * param rm contact string for the resource manager.
 * return the job service.
 */
public static JobService createJobService(
        Session session, URL rm)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        Timeout, NoSuccess {
    initializeFactory();
    return factory.doCreateJobService(session, rm);
}

/**
 * Creates a task that creates a job service.
 * param mode the task mode.
 * param session the session handle.
 * param rm contact string for the resource manager.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public static Task<JobService> createJobService(
        TaskMode mode, Session session, URL rm) throws NotImplemented {
    initializeFactory();
    return factory.doCreateJobService(mode, session, rm);
}

/**
 * Creates a job service.
 * param session the session handle.
 * return the job service.
 */
```

```
public static JobService createJobService(Session session)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        Timeout, NoSuccess {
    URL url;
    try {
        url = new URL("");
    } catch(Throwable e) {
        throw new SagaError("Should not happen", e);
    }
    initializeFactory();
    return factory.doCreateJobService(session, url);
}

/**
 * Creates a task that creates a job service, using a default contact string.
 * param mode the task mode.
 * param session the session handle.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public static Task<JobService> createJobService(
        TaskMode mode, Session session) throws NotImplemented {
    URL url;
    try {
        url = new URL("");
    } catch(Throwable e) {
        throw new SagaError("Should not happen", e);
    }
    initializeFactory();
    return factory.doCreateJobService(mode, session, url);
}

/**
 * Creates a job service, using the default session.
 * param rm contact string for the resource manager.
 * return the job service.
 */
public static JobService createJobService(URL rm)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        Timeout, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateJobService(session, rm);
}

/**
 * Creates a task that creates a job service, using the default session.
```

```
 * param mode the task mode.
 * param rm contact string for the resource manager.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 * exception NoSuccess is thrown when the default session could not be
 *     created.
 */
public static Task<JobService> createJobService(
        TaskMode mode, URL rm) throws NotImplemented, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateJobService(mode, session, rm);
}


/**
 * Creates a job service, using the default session and default contact
 * string.
 * return the job service.
 */
public static JobService createJobService()
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        Timeout, NoSuccess {
    URL url;
    try {
        url = new URL("");
    } catch(Throwable e) {
        throw new SagaError("Should not happen", e);
    }
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateJobService(session, url);
}


/**
 * Creates a task that creates a job service, using the default session
 * and default contact string.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 * exception NoSuccess is thrown when the default session could not be
 *     created.
 */
public static Task<JobService> createJobService(
        TaskMode mode) throws NotImplemented, NoSuccess {
    URL url;
    try {
        url = new URL("");
```

```
        } catch(Throwable e) {
            throw new SagaError("Should not happen", e);
        }
        Session session = SessionFactory.createSession();
        initializeFactory();
        return factory.doCreateJobService(mode, session, url);
    }
}
```

4.2   SAGA Name Spaces

Several SAGA packages share the notion of name spaces and operations on
these name spaces. In order to increase consistency in the API, these packages
share the same API paradigms. This section describes those paradigms, and
the interfaces that operate on various, hierarchical name spaces, such as used
in physical, virtual, and logical file systems, and in information systems.

4.2.1   NSDirectory

```
package org.ogf.saga.namespace;

import java.util.List;

import org.ogf.saga.URL;
import org.ogf.saga.error.AlreadyExists;
import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.DoesNotExist;
import org.ogf.saga.error.IncorrectState;
import org.ogf.saga.error.IncorrectURL;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;

/**
 * Represents a namespace entry that is a directory, and defines additional
 * methods for them.
 */
public interface NSDirectory extends NSEntry {

    /**
     * Changes the working directory.
     * param dir the directory to change to.
     */
    public void changeDir(URL dir)
        throws NotImplemented, IncorrectURL, AuthenticationFailed,
                         AuthorizationFailed, PermissionDenied, BadParameter,
                         IncorrectState, DoesNotExist, Timeout, NoSuccess;

    /**
     * Lists entries in the directory that match the specified pattern.
     * If the pattern is an empty string, all entries are listed.
```

```
    * The only allowed flag is DEREFERENCE.
    * param pattern name or pattern to list.
    * param flags defining the operation modus.
    * return the matching entries.
    */
   public List<URL> list(String pattern, int flags)
       throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
           PermissionDenied, BadParameter, IncorrectState, Timeout, NoSuccess,
           IncorrectURL;


   /**
    * Lists entries in the directory.
    * The only allowed flag is DEREFERENCE.
    * param flags defining the operation modus.
    * return the directory entries.
    */
   public List<URL> list(int flags)
       throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
           PermissionDenied, BadParameter, IncorrectState, Timeout, NoSuccess,
           IncorrectURL;


   /**
    * Lists entries in the directory that match the specified pattern.
    * If the pattern is an empty string, all entries are listed.
    * param pattern name or pattern to list.
    * return the matching entries.
    */
   public List<URL> list(String pattern)
       throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
           PermissionDenied, BadParameter, IncorrectState, Timeout, NoSuccess,
           IncorrectURL;


   /**
    * Lists entries in the directory.
    * return the directory entries.
    */
   public List<URL> list()
       throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
           PermissionDenied, BadParameter, IncorrectState, Timeout, NoSuccess,
           IncorrectURL;


   /**
    * Finds entries in the directory and below that match the specified
    * pattern.
    * If the pattern is an empty string, all entries are listed.
    * param pattern name or pattern to find.
    * param flags defining the operation modus.
    * return the matching entries.
    */
   public List<URL> find(String pattern, int flags)
```

```
      throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
          PermissionDenied, BadParameter, IncorrectState, Timeout, NoSuccess;

  /**
   * Finds entries in the directory and below that match the specified
   * pattern.
   * If the pattern is an empty string, all entries are listed.
   * param pattern name or pattern to find.
   * return the matching entries.
   */
  public List<URL> find(String pattern)
      throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
          PermissionDenied, BadParameter, IncorrectState, Timeout, NoSuccess;

  /**
   * Queries for the existence of an entry.
   * param name to be tested for existence.
   * return <code>true</code> if the name exists.
   */
  public boolean exists(URL name)
      throws NotImplemented, IncorrectURL, AuthenticationFailed,
          AuthorizationFailed, PermissionDenied, BadParameter,
          IncorrectState, Timeout, NoSuccess;

  /**
   * Tests the name for being a directory.
   * param name to be tested.
   * return <code>true</code> if the name represents a directory.
   */
  public boolean isDir(URL name)
      throws NotImplemented, IncorrectURL, DoesNotExist, AuthenticationFailed,
          AuthorizationFailed, PermissionDenied, BadParameter,
          IncorrectState, Timeout, NoSuccess;

  /**
   * Tests the name for being a namespace entry.
   * param name to be tested.
   * return <code>true</code> if the name represents a non-directory entry.
   */
  public boolean isEntry(URL name)
      throws NotImplemented, IncorrectURL, DoesNotExist, AuthenticationFailed,
          AuthorizationFailed, PermissionDenied, BadParameter,
          IncorrectState, Timeout, NoSuccess;

  /**
   * Tests the name for being a link.
   * param name to be tested.
   * return <code>true</code> if the name represents a link.
   */
  public boolean isLink(URL name)
```

```
    throws NotImplemented, IncorrectURL, DoesNotExist, AuthenticationFailed,
        AuthorizationFailed, PermissionDenied, BadParameter,
        IncorrectState, Timeout, NoSuccess;

/**
 * Returns the URL representing the link target.
 * param name the name of the link.
 * return the resolved name.
 */
public URL readLink(URL name)
    throws NotImplemented, IncorrectURL, DoesNotExist, AuthenticationFailed,
        AuthorizationFailed, PermissionDenied, BadParameter,
        IncorrectState, Timeout, NoSuccess;

// TODO: replace the next two methods by making NamespaceDirectory extend
// Iterable<URL>??? What to do then about the async versions?

/**
 * Obtains the number of entries in this directory.
 * return the number of entries.
 */
public int getNumEntries()
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectState, Timeout, NoSuccess;

/**
 * Gives the name of an entry in the directory based upon the
 * enumeration defined by getNumEntries().
 * param entry index of the entry to get.
 * return the name of the entry.
 * exception DoesNotExistException is thrown when the index is invalid.
 */
public URL getEntry(int entry)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectState, DoesNotExist,
        Timeout, NoSuccess;

/**
 * Copies the source entry to another part of the namespace.
 * param source name to copy.
 * param target name to copy to.
 * param flags defining the operation modus.
 */
public void copy(URL source, URL target, int flags)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectURL, BadParameter, IncorrectState,
        AlreadyExists, DoesNotExist, Timeout, NoSuccess;

/**
 * Copies the source entry to another part of the namespace.
```

```
 * param source name to copy.
 * param target name to copy to.
 */
public void copy(URL source, URL target)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectURL, BadParameter, IncorrectState,
        AlreadyExists, DoesNotExist, Timeout, NoSuccess;


/**
 * Copies the source entry to another part of the namespace. The source
 * may contain wildcards.
 * param source name to copy.
 * param target name to copy to.
 * param flags defining the operation modus.
 */
public void copy(String source, URL target, int flags)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectURL, BadParameter, IncorrectState,
        AlreadyExists, DoesNotExist, Timeout, NoSuccess;


/**
 * Copies the source entry to another part of the namespace. The source
 * may contain wildcards.
 * param source name to copy.
 * param target name to copy to.
 */
public void copy(String source, URL target)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectURL, BadParameter, IncorrectState,
        AlreadyExists, DoesNotExist, Timeout, NoSuccess;


/**
 * Creates a symbolic link from the specified target to the
 * specified source.
 * param source name to link to.
 * param target name of the link.
 * param flags defining the operation modus.
 */
public void link(URL source, URL target, int flags)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectURL, BadParameter, IncorrectState,
        AlreadyExists, DoesNotExist, Timeout, NoSuccess;


/**
 * Creates a symbolic link from the specified target to the
 * specified source.
 * param source name to link to.
 * param target name of the link.
 */
public void link(URL source, URL target)
```

```
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, IncorrectURL, BadParameter, IncorrectState,
            AlreadyExists, DoesNotExist, Timeout, NoSuccess;

/**
 * Creates a symbolic link from the specified target to the
 * specified source. The source may contain wildcards.
 * param source name to link to.
 * param target name of the link.
 * param flags defining the operation modus.
 */
public void link(String source, URL target, int flags)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectURL, BadParameter, IncorrectState,
        AlreadyExists, DoesNotExist, Timeout, NoSuccess;

/**
 * Creates a symbolic link from the specified target to the
 * specified source. The source may contain wildcards.
 * param source name to link to.
 * param target name of the link.
 */
public void link(String source, URL target)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectURL, BadParameter, IncorrectState,
        AlreadyExists, DoesNotExist, Timeout, NoSuccess;

/**
 * Renames the specified source to the specified target, or move the
 * specified source to the specified target if the target is a directory.
 * param source name to move.
 * param target name to move to.
 * param flags defining the operation modus.
 */
public void move(URL source, URL target, int flags)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectURL, BadParameter, IncorrectState,
        AlreadyExists, DoesNotExist, Timeout, NoSuccess;

/**
 * Renames the specified source to the specified target, or move the
 * specified source to the specified target if the target is a directory.
 * param source name to move.
 * param target name to move to.
 */
public void move(URL source, URL target)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectURL, BadParameter, IncorrectState,
        AlreadyExists, DoesNotExist, Timeout, NoSuccess;
```

```
/**
 * Renames the specified source to the specified target, or move the
 * specified source to the specified target if the target is a directory.
 * The source may contain wildcards.
 * param source name to move.
 * param target name to move to.
 * param flags defining the operation modus.
 */
public void move(String source, URL target, int flags)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectURL, BadParameter, IncorrectState,
        AlreadyExists, DoesNotExist, Timeout, NoSuccess;


/**
 * Renames the specified source to the specified target, or move the
 * specified source to the specified target if the target is a directory.
 * The source may contain wildcards.
 * param source name to move.
 * param target name to move to.
 */
public void move(String source, URL target)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectURL, BadParameter, IncorrectState,
        AlreadyExists, DoesNotExist, Timeout, NoSuccess;


/**
 * Removes the specified entry.
 * param target name to remove.
 * param flags defining the operation modus.
 */
public void remove(URL target, int flags)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectURL, BadParameter, IncorrectState,
        DoesNotExist, Timeout, NoSuccess;


/**
 * Removes the specified entry.
 * param target name to remove.
 */
public void remove(URL target)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectURL, BadParameter, IncorrectState,
        DoesNotExist, Timeout, NoSuccess;


/**
 * Removes the specified entry.
 * The target string may contain wildcards.
 * param target name to remove.
 * param flags defining the operation modus.
 */
```

```
    public void remove(String target, int flags)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, IncorrectURL, BadParameter, IncorrectState,
            DoesNotExist, Timeout, NoSuccess;


    /**
     * Removes the specified entry.
     * The target string may contain wildcards.
     * param target name to remove.
     */
    public void remove(String target)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, IncorrectURL, BadParameter, IncorrectState,
            DoesNotExist, Timeout, NoSuccess;


    /**
     * Creates a new directory.
     * param target directory to create.
     * param flags defining the operation modus.
     */
    public void makeDir(URL target, int flags)
        throws NotImplemented, IncorrectURL, AuthenticationFailed,
            AuthorizationFailed, PermissionDenied, BadParameter,
            IncorrectState, AlreadyExists, DoesNotExist, Timeout, NoSuccess;


    /**
     * Creates a new directory.
     * param target directory to create.
     */
    public void makeDir(URL target)
        throws NotImplemented, IncorrectURL, AuthenticationFailed,
            AuthorizationFailed, PermissionDenied, BadParameter,
            IncorrectState, AlreadyExists, DoesNotExist, Timeout, NoSuccess;


    /**
     * Creates a new <code>NamespaceDirectory</code> instance.
     * param name directory to open.
     * param flags defining the operation modus.
     * return the opened directory instance.
     */
    public NSDirectory openDir(URL name, int flags)
        throws NotImplemented, IncorrectURL,
            AuthenticationFailed, AuthorizationFailed, PermissionDenied,
            BadParameter, IncorrectState, AlreadyExists, DoesNotExist,
            Timeout, NoSuccess;


    /**
     * Creates a new <code>NamespaceDirectory</code> instance.
     * param name directory to open.
     * return the opened directory instance.
```

```
    */
  public NSDirectory openDir(URL name)
      throws NotImplemented, IncorrectURL,
          AuthenticationFailed, AuthorizationFailed, PermissionDenied,
          BadParameter, IncorrectState, AlreadyExists, DoesNotExist,
          Timeout, NoSuccess;


  /**
   * Creates a new <code>NamespaceEntry</code> instance.
   * param name entry to open.
   * param flags defining the operation modus.
   * return the opened entry instance.
   */
  public NSEntry open(URL name, int flags)
      throws NotImplemented, IncorrectURL,
          AuthenticationFailed, AuthorizationFailed, PermissionDenied,
          BadParameter, IncorrectState, AlreadyExists, DoesNotExist,
          Timeout, NoSuccess;


  /**
   * Creates a new <code>NamespaceEntry</code> instance.
   * param name entry to open.
   * return the opened entry instance.
   */
  public NSEntry open(URL name)
      throws NotImplemented, IncorrectURL,
          AuthenticationFailed, AuthorizationFailed, PermissionDenied,
          BadParameter, IncorrectState, AlreadyExists, DoesNotExist,
          Timeout, NoSuccess;


  /**
   * Allows the specified permissions for the specified id.
   * An id of "*" enables the permissions for all.
   * param target the entry affected.
   * param id the id.
   * param permissions the permissions to enable.
   * param flags the only allowed flags are RECURSIVE and DEREFERENCE.
   */
  public void permissionsAllow(URL target, String id, int permissions,
          int flags)
      throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
          PermissionDenied, IncorrectState, BadParameter, Timeout, NoSuccess;


  /**
   * Allows the specified permissions for the specified id.
   * An id of "*" enables the permissions for all.
   * param target the entry affected.
   * param id the id.
   * param permissions the permissions to enable.
   */
```

```
public void permissionsAllow(URL target, String id, int permissions)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectState, BadParameter, Timeout, NoSuccess;

/**
 * Allows the specified permissions for the specified id.
 * An id of "*" enables the permissions for all.
 * param target the entry affected.
 * param id the id.
 * param permissions the permissions to enable.
 * param flags the only allowed flags are RECURSIVE and DEREFERENCE.
 */
public void permissionsAllow(String target, String id, int permissions,
        int flags)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectState, BadParameter, Timeout, NoSuccess;

/**
 * Allows the specified permissions for the specified id.
 * An id of "*" enables the permissions for all.
 * param target the entry affected.
 * param id the id.
 * param permissions the permissions to enable.
 */
public void permissionsAllow(String target, String id, int permissions)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectState, BadParameter, Timeout, NoSuccess;

/**
 * Denies the specified permissions for the specified id.
 * An id of "*" disables the permissions for all.
 * param target the entry affected.
 * param id the id.
 * param permissions the permissions to disable.
 * param flags the only allowed flags are RECURSIVE and DEREFERENCE.
 */
public void permissionsDeny(URL target, String id, int permissions,
        int flags)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, Timeout, NoSuccess;

/**
 * Denies the specified permissions for the specified id.
 * An id of "*" disables the permissions for all.
 * param target the entry affected.
 * param id the id.
 * param permissions the permissions to disable.
 */
public void permissionsDeny(URL target, String id, int permissions)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
```

```
            PermissionDenied, BadParameter, Timeout, NoSuccess;

    /**
     * Denies the specified permissions for the specified id.
     * An id of "*" disables the permissions for all.
     * param target the entry affected.
     * param id the id.
     * param permissions the permissions to disable.
     * param flags the only allowed flags are RECURSIVE and DEREFERENCE.
     */
    public void permissionsDeny(String target, String id, int permissions,
            int flags)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, BadParameter, Timeout, NoSuccess;

    /**
     * Denies the specified permissions for the specified id.
     * An id of "*" disables the permissions for all.
     * param target the entry affected.
     * param id the id.
     * param permissions the permissions to disable.
     */
    public void permissionsDeny(String target, String id, int permissions)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, BadParameter, Timeout, NoSuccess;

    //
    // Task versions ...
    //

    /**
     * Creates a task that changes the working directory.
     * param mode the task mode.
     * param dir the directory to change to.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *      method is not implemented.
     */
    public Task changeDir(TaskMode mode, URL dir) throws NotImplemented;

    /**
     * Creates a task that lists entries in the directory that match
     * the specified pattern.
     * If the pattern is an empty string, all entries are listed.
     * The only allowed flag is DEREFERENCE.
     * param mode the task mode.
     * param pattern name or pattern to list.
     * param flags defining the operation modus.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
```

```
 *      method is not implemented.
 */
public Task<List<URL>> list(TaskMode mode, String pattern, int flags)
     throws NotImplemented;

/**
 * Creates a task that lists entries in the directory that match
 * the specified pattern.
 * If the pattern is an empty string, all entries are listed.
 * The only allowed flag is DEREFERENCE.
 * param mode the task mode.
 * param pattern name or pattern to list.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task<List<URL>> list(TaskMode mode, String pattern)
     throws NotImplemented;

/**
 * Creates a task that lists entries in the directory.
 * The only allowed flag is DEREFERENCE.
 * param mode the task mode.
 * param flags defining the operation modus.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task<List<URL>> list(TaskMode mode, int flags)
     throws NotImplemented;

/**
 * Creates a task that lists entries in the directory.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task<List<URL>> list(TaskMode mode) throws NotImplemented;

/**
 * Creates a task that finds entries in the directory and below that
 * match the specified pattern.
 * If the pattern is an empty string, all entries are listed.
 * param mode the task mode.
 * return the task.
 * param pattern name or pattern to find.
 * param flags defining the operation modus.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
```

```
 *      method is not implemented.
 */
public Task<List<URL>> find(TaskMode mode, String pattern, int flags)
    throws NotImplemented;

/**
 * Creates a task that finds entries in the directory and below that
 * match the specified pattern.
 * If the pattern is an empty string, all entries are listed.
 * param mode the task mode.
 * return the task.
 * param pattern name or pattern to find.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task<List<URL>> find(TaskMode mode, String pattern)
    throws NotImplemented;

/**
 * Creates a task that queries for the existence of an entry.
 * param mode the task mode.
 * param name to be tested for existence.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task<Boolean> exists(TaskMode mode, URL name)
    throws NotImplemented;

/**
 * Creates a task that tests the name for being a directory.
 * param mode the task mode.
 * param name to be tested.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task<Boolean> isDir(TaskMode mode, URL name)
    throws NotImplemented;

/**
 * Creates a task that tests the name for being a namespace entry.
 * param mode the task mode.
 * param name to be tested.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task<Boolean> isEntry(TaskMode mode, URL name)
```

```
        throws NotImplemented;

    /**
     * Creates a task that tests the name for being a link.
     * param mode the task mode.
     * param name to be tested.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task<Boolean> isLink(TaskMode mode, URL name)
        throws NotImplemented;

    /**
     * Creates a task that returns the URL representing the link target.
     * param mode the task mode.
     * param name the name of the link.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task<URL> readLink(TaskMode mode, URL name)
        throws NotImplemented;

    /**
     * Creates a task that obtains the number of entries in this directory.
     * param mode the task mode.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task<Integer> getNumEntries(TaskMode mode)
        throws NotImplemented;

    /**
     * Creates a task that gives the name of an entry in the directory based
     * upon the enumeration defined by getNumEntries().
     * param mode the task mode.
     * param entry index of the entry to get.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task<URL> getEntry(TaskMode mode, int entry)
        throws NotImplemented;

    /**
     * Creates a task that copies source the entry to another part of
     * the namespace.
     * param mode the task mode.
```

```
 * param source name to copy.
 * param target name to copy to.
 * param flags defining the operation modus.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task copy(TaskMode mode, URL source, URL target, int flags)
    throws NotImplemented;


/**
 * Creates a task that copies source the entry to another part of
 * the namespace.
 * param mode the task mode.
 * param source name to copy.
 * param target name to copy to.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task copy(TaskMode mode, URL source, URL target)
    throws NotImplemented;


/**
 * Creates a task that copies the source entry to another part of
 * the namespace. The source may contain wildcards.
 * param mode the task mode.
 * param source name to copy.
 * param target name to copy to.
 * param flags defining the operation modus.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task copy(TaskMode mode, String source, URL target, int flags)
    throws NotImplemented;


/**
 * Creates a task that copies the source entry to another part of
 * the namespace. The source may contain wildcards.
 * param mode the task mode.
 * param source name to copy.
 * param target name to copy to.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task copy(TaskMode mode, String source, URL target)
    throws NotImplemented;
```

```
/**
 * Creates a task that creates a symbolic link from the specified
 * target to the specified source.
 * param mode the task mode.
 * param source name to link to.
 * param target name of the link.
 * param flags defining the operation modus.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task link(TaskMode mode, URL source, URL target, int flags)
    throws NotImplemented;

/**
 * Creates a task that creates a symbolic link from the specified
 * target to the specified source.
 * param mode the task mode.
 * param source name to link to.
 * param target name of the link.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task link(TaskMode mode, URL source, URL target)
    throws NotImplemented;

/**
 * Creates a task that creates a symbolic link from the specified
 * target to the specified source. The source may contain wildcards.
 * param mode the task mode.
 * param source name to link to.
 * param target name of the link.
 * param flags defining the operation modus.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task link(TaskMode mode, String source, URL target, int flags)
    throws NotImplemented;

/**
 * Creates a task that creates a symbolic link from the specified
 * target to the specified source. The source may contain wildcards.
 * param mode the task mode.
 * param source name to link to.
 * param target name of the link.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
```

```
    */
    public Task link(TaskMode mode, String source, URL target)
        throws NotImplemented;

    /**
     * Creates a task that renames the specified source to the specified target,
     * or move the specified source to the specified target if the target is a
     * directory.
     * param mode the task mode.
     * param source name to move.
     * param target name to move to.
     * param flags defining the operation modus.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task move(TaskMode mode, URL source, URL target, int flags)
        throws NotImplemented;

    /**
     * Creates a task that renames the specified source to the specified target,
     * or move the specified source to the specified target if the target is a
     * directory.
     * param mode the task mode.
     * param source name to move.
     * param target name to move to.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task move(TaskMode mode, URL source, URL target)
        throws NotImplemented;

    /**
     * Creates a task that renames the specified source to the specified target,
     * or move the specified source to the specified target if the target is a
     * directory. The source may contain wildcards.
     * param mode the task mode.
     * param source name to move.
     * param target name to move to.
     * param flags defining the operation modus.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task move(TaskMode mode, String source, URL target, int flags)
        throws NotImplemented;

    /**
     * Creates a task that renames the specified source to the specified target,
```

```
 * or move the specified source to the specified target if the target is a
 * directory. The source may contain wildcards.
 * param mode the task mode.
 * param source name to move.
 * param target name to move to.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task move(TaskMode mode, String source, URL target)
    throws NotImplemented;


/**
 * Creates a task that removes the specified entry.
 * param mode the task mode.
 * param target name to remove.
 * param flags defining the operation modus.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task remove(TaskMode mode, URL target, int flags)
    throws NotImplemented;


/**
 * Creates a task that removes the specified entry.
 * param mode the task mode.
 * param target name to remove.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task remove(TaskMode mode, URL target)
    throws NotImplemented;


/**
 * Creates a task that removes the specified entry.
 * The target may contain wildcards.
 * param mode the task mode.
 * param target name to remove.
 * param flags defining the operation modus.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task remove(TaskMode mode, String target, int flags)
    throws NotImplemented;


/**
 * Creates a task that removes the specified entry.
```

```
 * The target may contain wildcards.
 * param mode the task mode.
 * param target name to remove.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task remove(TaskMode mode, String target)
    throws NotImplemented;


/**
 * Creates a task that creates a new directory.
 * param mode the task mode.
 * param target directory to create.
 * param flags defining the operation modus.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task makeDir(TaskMode mode, URL target, int flags)
    throws NotImplemented;


/**
 * Creates a task that creates a new directory.
 * param mode the task mode.
 * param target directory to create.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task makeDir(TaskMode mode, URL target)
    throws NotImplemented;


/**
 * Creates a task that creates a new <code>NamespaceDirectory</code>
 * instance.
 * param mode the task mode.
 * param name directory to open.
 * param flags defining the operation modus.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<NSDirectory> openDir(TaskMode mode, URL name,
        int flags)
    throws NotImplemented;


/**
 * Creates a task that creates a new <code>NamespaceDirectory</code>
 * instance.
```

```
 * param mode the task mode.
 * param name directory to open.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<NSDirectory> openDir(TaskMode mode, URL name)
    throws NotImplemented;


/**
 * Creates a task that creates a new <code>NamespaceEntry</code> instance.
 * param mode the task mode.
 * param name entry to open.
 * param flags defining the operation modus.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<NSEntry> open(TaskMode mode, URL name, int flags)
    throws NotImplemented;


/**
 * Creates a task that creates a new <code>NamespaceEntry</code> instance.
 * param mode the task mode.
 * param name entry to open.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<NSEntry> open(TaskMode mode, URL name)
    throws NotImplemented;


/**
 * Creates a task that enables the specified permissions for the
 * specified id.
 * An id of "*" enables the permissions for all.
 * param mode determines the initial state of the task.
 * param target the entry affected.
 * param id the id.
 * param permissions the permissions to enable.
 * param flags the only allowed flags are RECURSIVE and DEREFERENCE.
 * return the task object.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task permissionsAllow(TaskMode mode, URL target, String id,
        int permissions, int flags)
    throws NotImplemented;


/**
```

```
 * Creates a task that enables the specified permissions for the
 * specified id.
 * An id of "*" enables the permissions for all.
 * param mode determines the initial state of the task.
 * param target the entry affected.
 * param id the id.
 * param permissions the permissions to enable.
 * return the task object.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task permissionsAllow(TaskMode mode, URL target, String id,
        int permissions)
    throws NotImplemented;


/**
 * Creates a task that enables the specified permissions for the
 * specified id. The target may contain wildcards.
 * An id of "*" enables the permissions for all.
 * param mode determines the initial state of the task.
 * param target the entry affected.
 * param id the id.
 * param permissions the permissions to enable.
 * param flags the only allowed flags are RECURSIVE and DEREFERENCE.
 * return the task object.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task permissionsAllow(TaskMode mode, String target, String id,
        int permissions, int flags)
    throws NotImplemented;


/**
 * Creates a task that enables the specified permissions for the
 * specified id. The target may contain wildcards.
 * An id of "*" enables the permissions for all.
 * param mode determines the initial state of the task.
 * param target the entry affected.
 * param id the id.
 * param permissions the permissions to enable.
 * return the task object.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task permissionsAllow(TaskMode mode, String target, String id,
        int permissions)
    throws NotImplemented;

/**
 * Creates a task that disables the specified permissions for the
```

```
   * specified id.
   * An id of "*" disables the permissions for all.
   * param mode determines the initial state of the task.
   * param target the entry affected.
   * param id the id.
   * param permissions the permissions to disable.
   * param flags the only allowed flags are RECURSIVE and DEREFERENCE.
   * return the task object.
   * exception NotImplemented is thrown when the task version of this
   *     method is not implemented.
   */
  public Task permissionsDeny(TaskMode mode, URL target, String id,
          int permissions, int flags)
      throws NotImplemented;


  /**
   * Creates a task that disables the specified permissions for the
   * specified id.
   * An id of "*" disables the permissions for all.
   * param mode determines the initial state of the task.
   * param target the entry affected.
   * param id the id.
   * param permissions the permissions to disable.
   * return the task object.
   * exception NotImplemented is thrown when the task version of this
   *     method is not implemented.
   */
  public Task permissionsDeny(TaskMode mode, URL target, String id,
          int permissions)
      throws NotImplemented;


  /**
   * Creates a task that disables the specified permissions for the
   * specified id. The target may contain wildcards.
   * An id of "*" disables the permissions for all.
   * param mode determines the initial state of the task.
   * param target the entry affected.
   * param id the id.
   * param permissions the permissions to disable.
   * param flags the only allowed flags are RECURSIVE and DEREFERENCE.
   * return the task object.
   * exception NotImplemented is thrown when the task version of this
   *     method is not implemented.
   */
  public Task permissionsDeny(TaskMode mode, String target, String id,
          int permissions, int flags)
      throws NotImplemented;


  /**
   * Creates a task that disables the specified permissions for the
```

```
        * specified id. The target may contain wildcards.
        * An id of "*" disables the permissions for all.
        * param mode determines the initial state of the task.
        * param target the entry affected.
        * param id the id.
        * param permissions the permissions to disable.
        * return the task object.
        * exception NotImplemented is thrown when the task version of this
        *     method is not implemented.
        */
    public Task permissionsDeny(TaskMode mode, String target, String id,
              int permissions)
          throws NotImplemented;
}
```


4.2.2   NSEntry


```
package org.ogf.saga.namespace;

import org.ogf.saga.SagaObject;
import org.ogf.saga.URL;
import org.ogf.saga.error.AlreadyExists;
import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.DoesNotExist;
import org.ogf.saga.error.IncorrectState;
import org.ogf.saga.error.IncorrectURL;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;
import org.ogf.saga.permissions.Permissions;
import org.ogf.saga.task.Async;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;

/**
 * Defines methods that allow inspection and management of the entry.
 */
public interface NSEntry extends SagaObject, Async, Permissions {

    /**
     * Obtains the complete URL refering to the entry.
     * return the URL.
     */
    public URL getURL()
        throws NotImplemented, IncorrectState, Timeout, NoSuccess;
```

```
/**
 * Obtains the current working directory for the entry.
 * return the current working directory.
 */
public URL getCWD()
    throws NotImplemented, IncorrectState, Timeout, NoSuccess;


/**
 * Obtains the name part of the URL of this entry.
 * return the name part.
 */
public URL getName()
    throws NotImplemented, IncorrectState, Timeout, NoSuccess;


/**
 * Tests this entry for being a directory.
 * return true if the entry is a directory.
 */
public boolean isDir()
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, IncorrectState, Timeout, NoSuccess;


/**
 * Tests this entry for being a namespace entry. If this entry represents
 * a link or a directory, this method returns <code>false</code>, although
 * strictly speaking, directories and links are namespace entries as well.
 * return true if the entry is a namespace entry.
 */
public boolean isEntry()
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, IncorrectState, Timeout, NoSuccess;


/**
 * Tests this entry for being a link.
 * return true if the entry is a link.
 */
public boolean isLink()
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, IncorrectState, Timeout, NoSuccess;


/**
 * Returns the URL representing the link target. Resolves one link level
 * only.
 * return the link target.
 */
public URL readLink()
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, IncorrectState, Timeout, NoSuccess;
```

```
/**
 * Copies this entry to another part of the namespace.
 * param target the name to copy to.
 * param flags defining the operation modus.
 */
public void copy(URL target, int flags)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, IncorrectState, AlreadyExists,
        DoesNotExist, Timeout, NoSuccess, IncorrectURL;


/**
 * Copies this entry to another part of the namespace.
 * param target the name to copy to.
 */
public void copy(URL target)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, IncorrectState, AlreadyExists,
        DoesNotExist, Timeout, NoSuccess, IncorrectURL;


/**
 * Creates a symbolic link from the target to this entry.
 * param target the name that will have the symbolic link to this entry.
 * param flags defining the operation modus.
 */
public void link(URL target, int flags)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, IncorrectState, AlreadyExists,
        Timeout, NoSuccess, IncorrectURL;


/**
 * Creates a symbolic link from the target to this entry.
 * param target the name that will have the symbolic link to this entry.
 */
public void link(URL target)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, IncorrectState, AlreadyExists,
        Timeout, NoSuccess, IncorrectURL;


/**
 * Renames this entry to the target, or moves this entry to the target
 * if it is a directory.
 * param target the name to move to.
 * param flags defining the operation modus.
 */
public void move(URL target, int flags)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, IncorrectState, AlreadyExists,
        DoesNotExist, Timeout, NoSuccess, IncorrectURL;


/**
```

```
 * Renames this entry to the target, or moves this entry to the target
 * if it is a directory.
 * param target the name to move to.
 */
public void move(URL target)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, IncorrectState, AlreadyExists,
        DoesNotExist, Timeout, NoSuccess, IncorrectURL;


/**
 * Removes this entry and closes it.
 * param flags defining the operation modus.
 */
public void remove(int flags)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, IncorrectState, Timeout, NoSuccess;



/**
 * Removes this entry and closes it.
 */
public void remove()
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, IncorrectState, Timeout, NoSuccess;


/**
 * Closes this entry.
 * This is a non-blocking close. Any subsequent method invocation on the
 * object will throw an IncorrectState exception.
 */
public void close()
    throws NotImplemented, IncorrectState, NoSuccess;


/**
 * Closes this entry.
 * Any subsequent method invocation on the
 * object will throw an IncorrectState exception.
 * param timeoutInSeconds seconds to wait.
 */
public void close(float timeoutInSeconds)
    throws NotImplemented, IncorrectState, NoSuccess;


/**
 * Allows the specified permissions for the specified id.
 * An id of "*" enables the permissions for all.
 * param id the id.
 * param permissions the permissions to enable.
 * param flags the only allowed flags are RECURSIVE and DEREFERENCE.
 */
public void permissionsAllow(String id, int permissions,
```

```
            int flags)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, IncorrectState, BadParameter, Timeout, NoSuccess;

    /**
     * Denies the specified permissions for the specified id.
     * An id of "*" disables the permissions for all.
     * param id the id.
     * param permissions the permissions to disable.
     * param flags the only allowed flags are RECURSIVE and DEREFERENCE.
     */
    public void permissionsDeny(String id, int permissions,
            int flags)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            IncorrectState, PermissionDenied, BadParameter, Timeout, NoSuccess;

    //
    // Task versions ...
    //

    /**
     * Creates a task that obtains the complete URL pointing to the entry.
     * param mode the task mode.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task<URL> getURL(TaskMode mode)
        throws NotImplemented;

    /**
     * Creates a task that obtains a String representing the current working
     * directory for the entry.
     * param mode the task mode.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task<URL> getCWD(TaskMode mode)
        throws NotImplemented;

    /**
     * Creates a task that obtains the name part of the URL of this entry.
     * param mode the task mode.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task<URL> getName(TaskMode mode)
        throws NotImplemented;
```

```
/**
 * Creates a task that tests this entry for being a directory.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<Boolean> isDir(TaskMode mode) throws NotImplemented;

/**
 * Creates a task that tests this entry for being a namespace entry.
 * If this entry represents
 * a link or a directory, this method returns <code>false</code>, although
 * strictly speaking, directories and links are namespace entries as well.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<Boolean> isEntry(TaskMode mode) throws NotImplemented;

/**
 * Creates a task that tests this entry for being a link.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<Boolean> isLink(TaskMode mode) throws NotImplemented;

/**
 * Creates a task that returns the URL representing the link target.
 * Resolves one link level only.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<URL> readLink(TaskMode mode)
    throws NotImplemented;

/**
 * Creates a task that copies this entry to another part of the namespace.
 * param mode the task mode.
 * param target the name to copy to.
 * param flags defining the operation modus.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
```

```
  */
 public Task copy(TaskMode mode, URL target, int flags)
     throws NotImplemented;

 /**
  * Creates a task that copies this entry to another part of the namespace.
  * param mode the task mode.
  * param target the name to copy to.
  * return the task.
  * exception NotImplemented is thrown when the task version of this
  *     method is not implemented.
  */
 public Task copy(TaskMode mode, URL target) throws NotImplemented;

 /**
  * Creates a task that creates a symbolic link from the target to this
  * entry.
  * param mode the task mode.
  * param target the name that will have the symbolic link to this entry.
  * param flags defining the operation modus.
  * return the task.
  * exception NotImplemented is thrown when the task version of this
  *     method is not implemented.
  */
 public Task link(TaskMode mode, URL target, int flags)
     throws NotImplemented;

 /**
  * Creates a task that creates a symbolic link from the target to this
  * entry.
  * param mode the task mode.
  * param target the name that will have the symbolic link to this entry.
  * return the task.
  * exception NotImplemented is thrown when the task version of this
  *     method is not implemented.
  */
 public Task link(TaskMode mode, URL target) throws NotImplemented;

 /**
  * Creates a task that renames this entry to the target, or moves this
  * entry to the target if it is a directory.
  * param mode the task mode.
  * param target the name to move to.
  * param flags defining the operation modus.
  * return the task.
  * exception NotImplemented is thrown when the task version of this
  *     method is not implemented.
  */
 public Task move(TaskMode mode, URL target, int flags)
     throws NotImplemented;
```

```
/**
 * Creates a task that renames this entry to the target, or moves this
 * entry to the target if it is a directory.
 * param mode the task mode.
 * param target the name to move to.
  * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task move(TaskMode mode, URL target) throws NotImplemented;

/**
 * Creates a task that removes this entry and closes it.
 * param mode the task mode.
 * param flags defining the operation modus.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task remove(TaskMode mode, int flags)
    throws NotImplemented;

/**
 * Creates a task that removes this entry and closes it.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */

public Task remove(TaskMode mode) throws NotImplemented;

/**
 * Creates a task that closes this entry.
 * This is a non-blocking close. When the task is done,
 * any subsequent method invocation on the
 * object will throw an IncorrectState exception.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task close(TaskMode mode)
    throws NotImplemented;

/**
 * Creates a task that closes this entry.
 * When the task is done, any subsequent method invocation on the
 * object will throw an IncorrectState exception.
```

```
     * param mode the task mode.
     * param timeoutInSeconds seconds to wait.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task close(TaskMode mode, float timeoutInSeconds)
        throws NotImplemented;

    /**
     * Creates a task that enables the specified permissions for the
     * specified id.
     * An id of "*" enables the permissions for all.
     * param mode determines the initial state of the task.
     * param id the id.
     * param permissions the permissions to enable.
     * param flags the only allowed flags are RECURSIVE and DEREFERENCE.
     * return the task object.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task permissionsAllow(TaskMode mode, String id,
            int permissions, int flags)
        throws NotImplemented;

    /**
     * Creates a task that disables the specified permissions for the
     * specified id.
     * An id of "*" disables the permissions for all.
     * param mode determines the initial state of the task.
     * param id the id.
     * param permissions the permissions to disable.
     * param flags the only allowed flags are RECURSIVE and DEREFERENCE.
     * return the task object.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task permissionsDeny(TaskMode mode, String id,
            int permissions, int flags)
        throws NotImplemented;
}
```

4.2.3 NSFactory

```
package org.ogf.saga.namespace;

import org.ogf.saga.URL;
import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
```

```
import org.ogf.saga.error.AlreadyExists;
import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.DoesNotExist;
import org.ogf.saga.error.IncorrectURL;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;
import org.ogf.saga.session.Session;
import org.ogf.saga.session.SessionFactory;
import org.ogf.saga.task.TaskMode;
import org.ogf.saga.task.Task;

/**
 * Factory for objects from the namespace package.
 */
public abstract class NSFactory {

    private static NSFactory factory;

    private static synchronized void initializeFactory()
        throws NotImplemented {
        if (factory == null) {
            factory = ImplementationBootstrapLoader.createNamespaceFactory();
        }
    }

    /**
     * Creates a task that creates a namespace entry.
     * To be provided by the implementation.
     * param mode the task mode.
     * param session the session handle.
     * param name the initial working directory.
     * param flags the open mode.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    protected abstract Task<NSEntry> doCreateNSEntry(
            TaskMode mode, Session session, URL name, int flags)
        throws NotImplemented;

    /**
     * Creates a task that creates a namespace directory.
     * To be provided by the implementation.
     * param mode the task mode.
     * param session the session handle.
     * param name the initial working directory.
```

```
 * param flags the open mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
protected abstract Task<NSDirectory>
        doCreateNSDirectory(TaskMode mode, Session session, URL name,
                  int flags)
        throws NotImplemented;


/**
 * Creates a namespace entry. To be provided by the implementation.
 * param session the session handle.
 * param name the initial working directory.
 * param flags the open mode.
 * return the namespace entry.
 */
protected abstract NSEntry doCreateNSEntry(
        Session session, URL name, int flags)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, DoesNotExist, AlreadyExists, Timeout, NoSuccess;

/**
 * Creates a namespace directory. To be provided by the implementation.
 * param session the session handle.
 * param name the initial working directory.
 * param flags the open mode.
 * return the namespace directory.
 */
protected abstract NSDirectory doCreateNSDirectory(
        Session session, URL name, int flags)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, DoesNotExist, AlreadyExists, Timeout, NoSuccess;

/**
 * Creates a namespace entry.
 * param session the session handle.
 * param name the initial working directory.
 * param flags the open mode.
 * return the namespace entry.
 */
public static NSEntry createNSEntry(
        Session session, URL name, int flags)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, DoesNotExist, AlreadyExists, Timeout, NoSuccess {
    initializeFactory();
```

```
        return factory.doCreateNSEntry(session, name, flags);
    }

    /**
     * Creates a namespace entry.
     * param session the session handle.
     * param name the initial working directory.
     * return the namespace entry.
     */
    public static NSEntry createNSEntry(
            Session session, URL name)
        throws NotImplemented, IncorrectURL,
            AuthenticationFailed, AuthorizationFailed, PermissionDenied,
            BadParameter, DoesNotExist, AlreadyExists, Timeout, NoSuccess {
        initializeFactory();
        return factory.doCreateNSEntry(session, name, Flags.NONE.getValue());
    }

    /**
     * Creates a namespace directory.
     * param session the session handle.
     * param name the initial working directory.
     * param flags the open mode.
     * return the namespace entry.
     */
    public static NSDirectory createNSDirectory(
            Session session, URL name, int flags)
        throws NotImplemented, IncorrectURL,
            AuthenticationFailed, AuthorizationFailed, PermissionDenied,
            BadParameter, DoesNotExist, AlreadyExists, Timeout, NoSuccess {
        initializeFactory();
        return factory.doCreateNSDirectory(session, name, flags);
    }

    /**
     * Creates a namespace directory.
     * param session the session handle.
     * param name the initial working directory.
     * return the namespace entry.
     */
    public static NSDirectory createNSDirectory(
            Session session, URL name)
        throws NotImplemented, IncorrectURL,
            AuthenticationFailed, AuthorizationFailed, PermissionDenied,
            BadParameter, DoesNotExist, AlreadyExists, Timeout, NoSuccess {
        initializeFactory();
        return factory.doCreateNSDirectory(session, name, Flags.NONE.getValue());
    }

    /**
```

```
 * Creates a task that creates a namespace entry.
 * param mode the task mode.
 * param session the session handle.
 * param name the initial working directory.
 * param flags the open mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public static Task<NSEntry> createNSEntry(TaskMode mode,
        Session session, URL name, int flags)
    throws NotImplemented {
    initializeFactory();
    return factory.doCreateNSEntry(mode, session, name, flags);
}


/**
 * Creates a task that creates a namespace entry.
 * param mode the task mode.
 * param session the session handle.
 * param name the initial working directory.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public static Task<NSEntry> createNSEntry(TaskMode mode,
        Session session, URL name)
    throws NotImplemented {
    initializeFactory();
    return factory.doCreateNSEntry(mode, session, name, Flags.NONE.getValue());
}



/**
 * Creates a task that creates a namespace directory.
 * param mode the task mode.
 * param session the session handle.
 * param name the initial working directory.
 * param flags the open mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */

public static Task<NSDirectory> createNSDirectory(
        TaskMode mode, Session session, URL name, int flags)
    throws NotImplemented {
    initializeFactory();
    return factory.doCreateNSDirectory(mode, session, name, flags);
}
```

```
/**
 * Creates a task that creates a namespace directory.
 * param mode the task mode.
 * param session the session handle.
 * param name the initial working directory.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public static Task<NSDirectory> createNSDirectory(
        TaskMode mode, Session session, URL name)
    throws NotImplemented {
    initializeFactory();
    return factory.doCreateNSDirectory(mode, session, name, Flags.NONE.getValue());
}




/**
 * Creates a namespace entry using the default session.
 * param name the initial working directory.
 * param flags the open mode.
 * return the namespace entry.
 */
public static NSEntry createNSEntry(URL name, int flags)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, DoesNotExist, AlreadyExists, Timeout, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateNSEntry(session, name, flags);
}

/**
 * Creates a namespace entry using the default session.
 * param name the initial working directory.
 * return the namespace entry.
 */
public static NSEntry createNSEntry(URL name)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, DoesNotExist, AlreadyExists, Timeout, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateNSEntry(session, name, Flags.NONE.getValue());
}

/**
 * Creates a namespace directory using the default session.
```

```
 * param name the initial working directory.
 * param flags the open mode.
 * return the namespace entry.
 */
public static NSDirectory createNSDirectory(URL name, int flags)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, DoesNotExist, AlreadyExists, Timeout, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateNSDirectory(session, name, flags);
}


/**
 * Creates a namespace directory using the default session.
 * param name the initial working directory.
 * return the namespace entry.
 */
public static NSDirectory createNSDirectory(URL name)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, DoesNotExist, AlreadyExists, Timeout, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateNSDirectory(session, name, Flags.NONE.getValue());
}


/**
 * Creates a task that creates a namespace entry using the default session.
 * param mode the task mode.
  * param name the initial working directory.
 * param flags the open mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 * exception NoSuccess is thrown when the default session could not be
 *     created.
 */
public static Task<NSEntry> createNSEntry(TaskMode mode, URL name, int flags)
    throws NotImplemented, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateNSEntry(mode, session, name, flags);
}


/**
 * Creates a task that creates a namespace entry using the default session.
 * param mode the task mode.
 * param name the initial working directory.
 * return the task.
```

```
    * exception NotImplemented is thrown when the task version of this
    *     method is not implemented.
    * exception NoSuccess is thrown when the default session could not be
    *     created.
    */
   public static Task<NSEntry> createNSEntry(TaskMode mode, URL name)
       throws NotImplemented, NoSuccess {
       Session session = SessionFactory.createSession();
       initializeFactory();
       return factory.doCreateNSEntry(mode, session, name, Flags.NONE.getValue());
   }


   /**
    * Creates a task that creates a namespace directory using the default session.
    * param mode the task mode.
    * param name the initial working directory.
    * param flags the open mode.
    * return the task.
    * exception NotImplemented is thrown when the task version of this
    *     method is not implemented.
    * exception NoSuccess is thrown when the default session could not be
    *     created.
    */

   public static Task<NSDirectory> createNSDirectory(
           TaskMode mode, URL name, int flags)
       throws NotImplemented, NoSuccess {
       Session session = SessionFactory.createSession();
       initializeFactory();
       return factory.doCreateNSDirectory(mode, session, name, flags);
   }

   /**
    * Creates a task that creates a namespace directory using the default session.
    * param mode the task mode.
    * param name the initial working directory.
    * return the task.
    * exception NotImplemented is thrown when the task version of this
    *     method is not implemented.
    * exception NoSuccess is thrown when the default session could not be
    *     created.
    */
   public static Task<NSDirectory> createNSDirectory(TaskMode mode, URL name)
       throws NotImplemented, NoSuccess {
       Session session = SessionFactory.createSession();
       initializeFactory();
       return factory.doCreateNSDirectory(mode, session, name, Flags.NONE.getValue());
   }
}
```

4.2.4 Flags

```
package org.ogf.saga.namespace;

/**
 * Enumerates some flags for methods in this package.
 * Note: since enumerations cannot be extended, all flags are included here.
 * The SAGA specs gets away with this as it has no real enumeration type.
 * In Java, the values should all be of the same type, or else the file package
 * for instance cannot inherit from the namespace package.
 * These flags are meant to be or-ed together, resulting in an integer.
 * Java does not define arithmetic operators for enumerations,
 * so methods are added here to test for presence and or-ing.
 * For instance,
 * <br>
 * <code>
 * int flags = Flags.EXCL.or(Flags.READ.or(Flags.WRITE));
 * <br>
 * if (Flags.READ.isSet(flags)) ...
 * </code>
 */
public enum Flags {

    NONE (0),
    OVERWRITE (1),
    RECURSIVE (2),
    DEREFERENCE (4),
    CREATE (8),
    EXCL (16),
    LOCK (32),
    CREATEPARENTS (64),
    ALLNAMESPACEFLAGS(1|2|4|8|16|32|64),
    TRUNCATE(128),
    APPEND(256),
    READ(512),
    WRITE(1024),
    READWRITE(512|1024),
    BINARY(2048),
    ALLLOGICALFILEFLAGS(512|1024),
    ALLFILEFLAGS(128|256|512|1024|2048);

    private int value;

    Flags(int value) {
        this.value = value;
    }

    /**
     * Returns the integer value of this enumeration literal.
```

```
 * return the integer value.
 */
public int getValue() {
    return value;
}

/**
 * Returns the result of or-ing this flag into an integer.
 * param val the value to OR this enumeration value into.
 * return the result of or-ing this flag into the integer parameter.
 */
public int or(int val) {
    return val | value;
}

/**
 * Returns the result of or-ing this flag into another.
 * param val the value to OR this enumeration value into.
 * return the result of or-ing this flag into the integer parameter.
 */
public int or(Flags val) {
    return val.value | value;
}

/**
 * Tests for the presence of this flag in the specified value.
 * param val the value.
 * return <code>true</code> if this flag is present.
 */
public boolean isSet(int val) {
    if (value == val) {
        // Also tests for 0 (NONE) which is assumed to be set only when
        // no other values are set.
        return true;
    }
    return (val & value) != 0;
}
}
```

4.3   SAGA File Management

The ability to access the contents of files regardless of their location is central to many of the SAGA use cases. This section addresses the most common operations detailed in these use cases.

It is important to note that interactions with files as opaque entities (i.e. as entries in file name spaces) are covered by the `namespace` package. The classes presented here supplement the `namespace` package with operations for the reading and writing of the *contents* of files. For all methods, the descriptions and notes of the equivalent methods in the `namespace` package apply if available, unless noted here otherwise.

Earlier experience with JavaGAT [10] has shown that having implementations of the Java streams `java.io.InputStream` and `java.io.OutputStream` is very much appreciated by Java application programmers, since these are the types on which most Java I/O is based. Therefore, it was decided to add specifications for `FileInputStream` and `FileOutputStream` to the file package. The `file` class as specified in the SAGA specifications is also specified in the Java language bindings. Of course, implementations and factories may throw the `NotImplemented` exception when methods or complete classes cannot be implemented.

4.3.1   Directory

```
package org.ogf.saga.file;

import org.ogf.saga.URL;
import org.ogf.saga.error.AlreadyExists;
import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.DoesNotExist;
import org.ogf.saga.error.IncorrectState;
import org.ogf.saga.error.IncorrectURL;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;
import org.ogf.saga.namespace.NSDirectory;
import org.ogf.saga.task.TaskMode;
import org.ogf.saga.task.Task;

/**
 * A Directory instance represents an open directory.
 */
public interface Directory extends NSDirectory {
```

```
// Inspection methods

/**
 * Returns the number of bytes in the specified file.
 * param name name of file to inspect.
 * param flags mode for operation.
 * return the size.
 */
public long getSize(URL name, int flags)
    throws NotImplemented, IncorrectURL, AuthenticationFailed,
        AuthorizationFailed, PermissionDenied, BadParameter,
        IncorrectState, DoesNotExist, Timeout, NoSuccess;

/**
 * Returns the number of bytes in the specified file.
 * param name name of file to inspect.
 * return the size.
 */
public long getSize(URL name)
    throws NotImplemented, IncorrectURL, AuthenticationFailed,
        AuthorizationFailed, PermissionDenied, BadParameter,
        IncorrectState, DoesNotExist, Timeout, NoSuccess;

/**
 * Tests the name for being a directory entry.
 * Is an alias for {link NSDirectory#isEntry}.
 * param name to be tested.
 * return <code>true</code> if the name represents a non-directory entry.
 */
public boolean isFile(URL name)
    throws NotImplemented, IncorrectURL, DoesNotExist, AuthenticationFailed,
        AuthorizationFailed, PermissionDenied, BadParameter,
        IncorrectState, Timeout, NoSuccess;

// openDirectory and openFile: names changed with respect
// to specs  because of Java restriction: cannot redefine methods with
// just a different return type.
// Thus, they don't hide the methods in NamespaceDirectory, but then,
// the ones in the SAGA spec don't either, because they have different
// out parameters.

/**
 * Creates a new <code>Directory</code> instance.
 * param name directory to open.
 * param flags defining the operation modus.
 * return the opened directory instance.
 */
public Directory openDirectory(URL name, int flags)
    throws NotImplemented, IncorrectURL,
```

```
                    AuthenticationFailed, AuthorizationFailed, PermissionDenied,
                    BadParameter, IncorrectState, AlreadyExists, DoesNotExist,
                    Timeout, NoSuccess;

     /**
      * Creates a new <code>Directory</code> instance.
      * param name directory to open.
      * return the opened directory instance.
      */
     public Directory openDirectory(URL name)
         throws NotImplemented, IncorrectURL,
             AuthenticationFailed, AuthorizationFailed, PermissionDenied,
             BadParameter, IncorrectState, AlreadyExists, DoesNotExist,
             Timeout, NoSuccess;

     /**
      * Creates a new <code>File</code> instance.
      * param name file to open.
      * param flags defining the operation modus.
      * return the opened file instance.
      */
     public File openFile(URL name, int flags)
         throws NotImplemented, IncorrectURL,
             AuthenticationFailed, AuthorizationFailed, PermissionDenied,
             BadParameter, IncorrectState, AlreadyExists, DoesNotExist,
             Timeout, NoSuccess;

     /**
      * Creates a new <code>File</code> instance.
      * param name file to open.
      * return the opened file instance.
      */
     public File openFile(URL name)
         throws NotImplemented, IncorrectURL,
             AuthenticationFailed, AuthorizationFailed, PermissionDenied,
             BadParameter, IncorrectState, AlreadyExists, DoesNotExist,
             Timeout, NoSuccess;

     /**
      * Creates a new <code>FileInputStream</code> instance.
      * param name file to open.
      * return the input stream.
      */
     public FileInputStream openFileInputStream(URL name)
         throws NotImplemented, IncorrectURL,
             AuthenticationFailed, AuthorizationFailed, PermissionDenied,
             BadParameter, IncorrectState, AlreadyExists, DoesNotExist,
             Timeout, NoSuccess;

     /**
```

```
 * Creates a new <code>FileOutputStream</code> instance.
 * param name file to open.
 * return the output stream.
 */
public FileOutputStream openFileOutputStream(URL name)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, IncorrectState, AlreadyExists, DoesNotExist,
        Timeout, NoSuccess;

/**
 * Creates a new <code>FileOutputStream</code> instance.
 * param name file to open.
 * param append when set, the stream appends to the file.
 * return the output stream.
 */
public FileOutputStream openFileOutputStream(URL name, boolean append)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, IncorrectState, AlreadyExists, DoesNotExist,
        Timeout, NoSuccess;

//
// Task versions
//

/**
 * Creates a task that retrieves the number of bytes in the specified file.
 * param mode the task mode.
 * param name name of file to inspect.
 * param flags mode for operation.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<Long> getSize(TaskMode mode, URL name, int flags)
    throws NotImplemented;

/**
 * Creates a task that retrieves the number of bytes in the specified file.
 * param mode the task mode.
 * param name name of file to inspect.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<Long> getSize(TaskMode mode, URL name)
    throws NotImplemented;

/**
```

```
 * Creates a task that tests the name for being a directory entry.
 * Is an alias for {link NSDirectory#isEntry}.
 * param mode the task mode.
 * param name to be tested.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task<Boolean> isFile(TaskMode mode, URL name)
    throws NotImplemented;


/**
 * Creates a task that creates a new <code>Directory</code> instance.
 * param mode the task mode.
 * param name directory to open.
 * param flags defining the operation modus.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task<Directory> openDirectory(TaskMode mode, URL name,
        int flags)
    throws NotImplemented;


/**
 * Creates a task that creates a new <code>Directory</code> instance.
 * param mode the task mode.
 * param name directory to open.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task<Directory> openDirectory(TaskMode mode, URL name)
    throws NotImplemented;


/**
 * Creates a task that creates a new <code>File</code> instance.
 * param mode the task mode.
 * param name file to open.
 * param flags defining the operation modus.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task<File> openFile(TaskMode mode, URL name, int flags)
    throws NotImplemented;


/**
 * Creates a task that creates a new <code>File</code> instance.
 * param mode the task mode.
```

```
    * param name file to open.
    * return the task.
    * exception NotImplemented is thrown when the task version of this
    *     method is not implemented.
    */
  public Task<File> openFile(TaskMode mode, URL name)
      throws NotImplemented;

  /**
   * Creates a task that creates a new <code>FileInputStream</code> instance.
   * param mode the task mode.
   * param name file to open.
   * return the task.
   * exception NotImplemented is thrown when the task version of this
   *     method is not implemented.
   */
  public Task<FileInputStream> openFileInputStream(TaskMode mode, URL name)
      throws NotImplemented;

  /**
   * Creates a task that creates a new <code>FileOutputStream</code> instance.
   * param mode the task mode.
   * param name file to open.
   * return the task.
   * exception NotImplemented is thrown when the task version of this
   *     method is not implemented.
   */
  public Task<FileOutputStream> openFileOutputStream(TaskMode mode, URL name)
      throws NotImplemented;

  /**
   * Creates a task that creates a new <code>FileOutputStream</code> instance.
   * param mode the task mode.
   * param name file to open.
   * param append when set, the file is opened for appending.
   * return the task.
   * exception NotImplemented is thrown when the task version of this
   *     method is not implemented.
   */
  public Task<FileOutputStream> openFileOutputStream(TaskMode mode, URL name, boolean append)
      throws NotImplemented;

}
```

4.3.2   File

```
package org.ogf.saga.file;
```

```
import java.io.IOException;
import java.util.List;

import org.ogf.saga.buffer.Buffer;
import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.IncorrectState;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;
import org.ogf.saga.namespace.NSEntry;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;

/**
 * The File interface represents an open file descriptor for reads/writes on a
 * physical file.
 * Errors result in an IOException, not a POSIX error code.
 */
public interface File extends NSEntry {

    // Inspection

    /**
     * Returns the number of bytes in the file.
     *
     * return the size.
     */
    public long getSize()
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, IncorrectState, Timeout, NoSuccess;

    // POSIX-like I/O

    /**
     * Reads up to <code>len</code> bytes from the file into the buffer.
     * Returns the number of bytes read, or 0 at end-of-file.
     * Note: this call is blocking. The async version can be used
     * to implement non-blocking reads.
     *
     * param buffer the buffer to read data into.
     * param len the number of bytes to be read.
     * return the number of bytes read.
     */
    public int read(Buffer buffer, int len)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, BadParameter,
            IncorrectState, Timeout, NoSuccess, IOException;
```

```
    /**
     * Reads up to <code>len</code> bytes from the file into the buffer,
     * at the specified <code>offset</code>.
     * Returns the number of bytes read, or 0 at end-of-file.
     * Note: this call is blocking. The async version can be used
     * to implement non-blocking reads.
     *
     * param buffer the buffer to read data into.
     * param offset ths offset in the buffer.
     * param len the number of bytes to be read.
     * return the number of bytes read.
     */
    public int read(Buffer buffer, int offset, int len)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, BadParameter,
            IncorrectState, Timeout, NoSuccess, IOException;


    /**
     * Reads up to the buffer's size from the file into the buffer.
     * Returns the number of bytes read, or 0 at end-of-file.
     * Note: this call is blocking. The async version can be used
     * to implement non-blocking reads.
     *
     * param buffer the buffer to read data into.
     * return the number of bytes read.
     */
    public int read(Buffer buffer)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, BadParameter,
            IncorrectState, Timeout, NoSuccess, IOException;

    /**
     * Writes up to <code>len</code> bytes from the buffer
     * at the specified buffer <code>offset</code> to the file
     * at the current file position.
     * Returns the number of bytes written.
     *
     * param buffer the buffer to write data from.
     * param offset the buffer offset.
     * param len the number of bytes to be written.
     * return the number of bytes written.
     */
    public int write(Buffer buffer, int offset, int len)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, BadParameter,
            IncorrectState, Timeout, NoSuccess, IOException;

    /**
```

```
 * Writes up to <code>len</code> bytes from the buffer to the file
 * at the current file position.
 * Returns the number of bytes written.
 *
 * param buffer the buffer to write data from.
 * param len the number of bytes to be written.
 * return the number of bytes written.
 */
public int write(Buffer buffer, int len)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter,
        IncorrectState, Timeout, NoSuccess, IOException;

/**
 * Writes up to the buffer's size bytes from the buffer to the file
 * at the current file position.
 * Returns the number of bytes written.
 *
 * param buffer the buffer to write data from.
 * return the number of bytes written.
 */
public int write(Buffer buffer)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter,
        IncorrectState, Timeout, NoSuccess, IOException;


/**
 * Repositions the current file position as requested.
 *
 * param offset offset in bytes to move pointer.
 * param whence determines from where the offset is relative.
 * return the position after the seek.
 */
public long seek(long offset, SeekMode whence)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectState, Timeout, NoSuccess,
        IOException;

// Scattered I/O

/**
 * Gather/scatter read.
 *
 * param iovecs array of IOVecs determining how much to read and where
 * to store it.
 */
public void readV(IOVec[] iovecs)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter,
```

```
                IncorrectState, Timeout, NoSuccess, IOException;

    /**
     * Gather/scatter write.
     *
     * param iovecs array of IOVecs determining how much to write and where
     * to obtain the data from.
     */
    public void writeV(IOVec[] iovecs)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, BadParameter,
            IncorrectState, Timeout, NoSuccess, IOException;

    // Pattern-based I/O

    /**
     * Determines the storage size required for a pattern I/O operation.
     * param pattern to determine size for.
     * return the size.
     */
    public int sizeP(String pattern)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            IncorrectState, PermissionDenied, BadParameter, Timeout, NoSuccess;

    /**
     * Pattern-based read.
     *
     * param pattern specification for the read operation.
     * param buffer to store data into.
     * return number of succesfully read bytes.
     */
    public int readP(String pattern, Buffer buffer)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, BadParameter, IncorrectState,
            Timeout, NoSuccess, IOException;

    /**
     * Pattern-based write.
     *
     * param pattern specification for the write operation.
     * param buffer to be written.
     * return number of succesfully written bytes.
     */
    public int writeP(String pattern, Buffer buffer)
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, BadParameter, IncorrectState,
            Timeout, NoSuccess, IOException;

    // extended I/O
```

```
/**
 * Lists the extended modes available in this implementation and/or on
 * the server side.
 * return list of available modes.
 */
public List<String> modesE()
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, IncorrectState, Timeout, NoSuccess;


/**
 * Determines the storage size required for an extended I/O operation.
 * param emode extended mode to use.
 * param spec to determine size for.
 * return the size.
 */
public int sizeE(String emode, String spec)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        IncorrectState, PermissionDenied, BadParameter, Timeout, NoSuccess;


/**
 * Extended read.
 *
 * param emode extended mode to use.
 * param spec specification of read operation.
 * param buffer to store the data read.
 * return the number of successfully read bytes.
 */
public int readE(String emode, String spec, Buffer buffer)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, IncorrectState, Timeout,
        NoSuccess, IOException;


/**
 * Extended write.
 *
 * param emode extended mode to use.
 * param spec specification of write operation.
 * param buffer data to write.
 * return the number of successfully written bytes.
 */
public int writeE(String emode, String spec, Buffer buffer)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, IncorrectState, Timeout,
        NoSuccess, IOException;


//
// Task versions ..
//

// Inspection
```

```
/**
 * Creates a task that obtains the number of bytes in the file.
 * This is the task version.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<Long> getSize(TaskMode mode)
    throws NotImplemented;


// POSIX-like I/O


/**
 * Creates a task that reads up to <code>len</code> bytes from the file
 * into the buffer at the specified buffer <code>offset</code>.
 * The number returned by the task is the number of bytes read, or 0 at
 * end-of-file.
 * param mode the task mode.
 * param offset the buffer offset.
 * param len the number of bytes to be read.
 * param buffer the buffer to read data into.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */

public Task<Integer> read(TaskMode mode, Buffer buffer, int offset, int len)
    throws NotImplemented;
/**
 * Creates a task that reads up to <code>len</code> bytes from the file
 * into the buffer.
 * The number returned by the task is the number of bytes read, or 0 at
 * end-of-file.
 * param mode the task mode.
 * param len the number of bytes to be read.
 * param buffer the buffer to read data into.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<Integer> read(TaskMode mode, Buffer buffer, int len)
    throws NotImplemented;


/**
 * Creates a task that reads up to the buffer's size bytes from the file
 * into the buffer.
 * The number returned by the task is the number of bytes read, or 0 at
 * end-of-file.
```

```
 * param mode the task mode.
 * param buffer the buffer to read data into.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<Integer> read(TaskMode mode, Buffer buffer)
    throws NotImplemented;

/**
 * Creates a task that writes up to <code>len</code> bytes from the buffer
 * at the specified buffer <code>offset</code>
 * to the file at the current file position.
 * The number returned by the task is the number of bytes written.
 * param mode the task mode.
 * param offset the buffer offset.
 * param len the number of bytes to be written.
 * param buffer the buffer to write data from.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<Integer> write(TaskMode mode, Buffer buffer, int offset, int len)
    throws NotImplemented;

/**
 * Creates a task that writes up to <code>len</code> bytes from the buffer
 * to the file at the current file position.
 * The number returned by the task is the number of bytes written.
 * param mode the task mode.
 * param len the number of bytes to be written.
 * param buffer the buffer to write data from.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<Integer> write(TaskMode mode, Buffer buffer, int len)
    throws NotImplemented;

/**
 * Creates a task that writes up to the buffer's size bytes from the buffer
 * to the file at the current file position.
 * The number returned by the task is the number of bytes written.
 * param mode the task mode.
 * param buffer the buffer to write data from.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<Integer> write(TaskMode mode, Buffer buffer)
```

```
      throws NotImplemented;

  /**
   * Creates a task that repositions the current file position as requested.
   * The number returned by the task is the new file position.
   * param mode the task mode.
   * param offset offset in bytes to move pointer.
   * param whence determines from where the offset is relative.
   * return the task.
   * exception NotImplemented is thrown when the task version of this
   *     method is not implemented.
   */
  public Task<Long> seek(TaskMode mode, long offset, SeekMode whence)
      throws NotImplemented;


  // Scattered I/O

  /**
   * Creates a task that does a gather/scatter read.
   * param mode the task mode.
   * param iovecs array of IOVecs determining how much to read and where
   * to store it.
   * return the task.
   * exception NotImplemented is thrown when the task version of this
   *     method is not implemented.
   */
  public Task readV(TaskMode mode, IOVec[] iovecs)
      throws NotImplemented;


  /**
   * Creates a task that does a gather/scatter write.
   * param mode the task mode.
   * param iovecs array of IOVecs determining how much to write and where
   * to obtain the data from.
   * return the task.
   * exception NotImplemented is thrown when the task version of this
   *     method is not implemented.
   */
  public Task writeV(TaskMode mode, IOVec[] iovecs)
      throws NotImplemented;


  // Pattern-based I/O

  /**
   * Creates a task that determines the storage size required for a
   * pattern I/O operation.
   * param mode the task mode.
   * param pattern to determine size for.
   * return the task.
   * exception NotImplemented is thrown when the task version of this
```

```
 *      method is not implemented.
 */
public Task<Integer> sizeP(TaskMode mode, String pattern)
    throws NotImplemented;


/**
 * Creates a task that does a pattern-based read.
 * param mode the task mode.
 * param pattern specification for the read operation.
 * param buffer to store data into.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task<Integer> readP(TaskMode mode, String pattern, Buffer buffer)
    throws NotImplemented;


/**
 * Creates a task that does a pattern-based write.
 * param mode the task mode.
 * param pattern specification for the write operation.
 * param buffer to be written.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task<Integer> writeP(TaskMode mode, String pattern, Buffer buffer)
    throws NotImplemented;


// extended I/O


/**
 * Creates a task that lists the extended modes available in this
 * implementation and/or on the server side.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public Task<List<String>> modesE(TaskMode mode)
    throws NotImplemented;


/**
 * Creates a task that determines the storage size required for an
 * extended I/O operation.
 * param mode the task mode.
 * param emode extended mode to use.
 * param spec to determine size for.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
```

```
    *     method is not implemented.
    */
  public Task<Integer> sizeE(TaskMode mode, String emode, String spec)
      throws NotImplemented;

  /**
   * Creates a task for an extended read.
   * param mode the task mode.
   * param emode extended mode to use.
   * param spec specification of read operation.
   * param buffer to store the data read.
   * return the task.
   * exception NotImplemented is thrown when the task version of this
   *     method is not implemented.
   */
  public Task<Integer> readE(TaskMode mode, String emode, String spec,
          Buffer buffer)
      throws NotImplemented;

  /**
   * Creates a task for an extended write.
   * param mode the task mode.
   * param emode extended mode to use.
   * param spec specification of write operation.
   * param buffer data to write.
   * return the task.
   * exception NotImplemented is thrown when the task version of this
   *     method is not implemented.
   */
  public Task<Integer> writeE(TaskMode mode, String emode, String spec,
          Buffer buffer)
      throws NotImplemented;
}
```

4.3.3 IOVec

```
package org.ogf.saga.file;

import org.ogf.saga.buffer.Buffer;
import org.ogf.saga.error.BadParameter;

/**
 * Extends the <code>Buffer</code> interface with lenIn, lenOut, and
 * offset attributes.
 */
public interface IOVec extends Buffer {

    /**
```

```
 * Sets the lenIn attribute.
 * param len the value for the attribute.
 */
void setLenIn(int len)
    throws BadParameter;

/**
 * Retrieves the current value of the lenIn attribute.
 * return the lenIn value.
 */
int getLenIn();

/**
 * Retrieves the current value of the lenOut attribute.
 * return the lenOut value.
 */
int getLenOut();

/**
 * Sets the offset attribute.
 * param offset the value for the attribute.
 */
void setOffset(int offset)
    throws BadParameter;

/**
 * Retrieves the current value of the offset attribute.
 * return the offset value.
 */
int getOffset();
}
```

4.3.4   FileFactory

```
package org.ogf.saga.file;

import org.ogf.saga.URL;
import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.AlreadyExists;
import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.DoesNotExist;
import org.ogf.saga.error.IncorrectURL;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;
```

```
import org.ogf.saga.namespace.Flags;
import org.ogf.saga.session.Session;
import org.ogf.saga.session.SessionFactory;
import org.ogf.saga.task.TaskMode;
import org.ogf.saga.task.Task;

/**
 * Factory for objects from the namespace package.
 */
public abstract class FileFactory {

    private static FileFactory factory;

    private static synchronized void initializeFactory() {
        if (factory == null) {
            factory = ImplementationBootstrapLoader.createFileFactory();
        }
    }

    /**
     * Creates an IOVec. To be provided by the implementation.
     * param data data to be used.
     * param lenIn number of bytes to read/write on readV/writeV.
     * return the IOVec.
     */
    protected abstract IOVec doCreateIOVec(byte[] data, int lenIn)
        throws BadParameter, NoSuccess, NotImplemented;

    /**
     * Creates an IOVec. To be provided by the implementation.
     * param size size of data to be used.
     * param lenIn number of bytes to read/write on readV/writeV.
     * return the IOVec.
     */
    protected abstract IOVec doCreateIOVec(int size, int lenIn)
        throws BadParameter, NoSuccess, NotImplemented;

    /**
     * Creates an IOVec. To be provided by the implementation.
     * param data data to be used.
     * return the IOVec.
     */
    protected abstract IOVec doCreateIOVec(byte[] data)
        throws BadParameter, NoSuccess, NotImplemented;

    /**
     * Creates an IOVec. To be provided by the implementation.
     * param size size of data to be used.
     * return the IOVec.
     */
```

```
    protected abstract IOVec doCreateIOVec(int size)
        throws BadParameter, NoSuccess, NotImplemented;


/**
 * Creates a File. To be provided by the implementation.
 * param session the session handle.
 * param name location of the file.
 * param flags the open mode.
 * return the file instance.
 */
protected abstract File doCreateFile(Session session, URL name, int flags)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, AlreadyExists, DoesNotExist,
        Timeout, NoSuccess;


/**
 * Creates a FileInputStream. To be provided by the implementation.
 * param session the session handle.
 * param name location of the file.
 * return the FileInputStream instance.
 */
protected abstract FileInputStream doCreateFileInputStream(Session session, URL name)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, AlreadyExists, DoesNotExist,
        Timeout, NoSuccess;


/**
 * Creates a FileOutputStream. To be provided by the implementation.
 * param session the session handle.
 * param name location of the file.
 * param append set when the file is opened for appending.
 * return the FileOutputStream instance.
 */
protected abstract FileOutputStream doCreateFileOutputStream(Session session,
        URL name, boolean append)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, AlreadyExists, DoesNotExist,
        Timeout, NoSuccess;


/**
 * Creates a Directory. To be provided by the implementation.
 * param session the session handle.
 * param name location of directory.
 * param flags the open mode.
 * return the directory instance.
 */
protected abstract Directory doCreateDirectory(Session session, URL name,
```

```
          int flags)
       throws NotImplemented, IncorrectURL,
          AuthenticationFailed, AuthorizationFailed, PermissionDenied,
          BadParameter, AlreadyExists, DoesNotExist, Timeout, NoSuccess;

   /**
    * Creates a Task that creates a File. To be provided by the implementation.
    * param mode the task mode.
    * param session the session handle.
    * param name location of the file.
    * param flags the open mode.
    * return the task.
    * exception NotImplemented is thrown when the task version of this
    *      method is not implemented.
    */
   protected abstract Task<File> doCreateFile(TaskMode mode,
          Session session, URL name, int flags)
       throws NotImplemented;

   /**
    * Creates a Task that creates a FileInputStream. To be provided by the implementation.
    * param mode the task mode.
    * param session the session handle.
    * param name location of the file.
    * return the task.
    * exception NotImplemented is thrown when the task version of this
    *      method is not implemented.
    */
   protected abstract Task<FileInputStream> doCreateFileInputStream(TaskMode mode,
          Session session, URL name)
       throws NotImplemented;

   /**
    * Creates a Task that creates a FileOutputStream. To be provided by the implementation.
    * param mode the task mode.
    * param session the session handle.
    * param name location of the file.
    * param append when set, the file is opened for appending.
    * return the task.
    * exception NotImplemented is thrown when the task version of this
    *      method is not implemented.
    */
   protected abstract Task<FileOutputStream> doCreateFileOutputStream(TaskMode mode,
          Session session, URL name, boolean append)
       throws NotImplemented;

   /**
    * Creates a Task that creates a Directory.
    * To be provided by the implementation.
    * param mode the task mode.
```

```
 * param session the session handle.
 * param name location of directory.
 * param flags the open mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
protected abstract Task<Directory> doCreateDirectory(
        TaskMode mode, Session session, URL name, int flags)
    throws NotImplemented;

/**
 * Creates an IOVec.
 * param data data to be used.
 * param lenIn number of bytes to read/write on readV/writeV.
 * return the IOVec.
 * throws NotImplemented
 */
public static IOVec createIOVec(byte[] data, int lenIn)
    throws BadParameter, NoSuccess, NotImplemented {
    initializeFactory();
    return factory.doCreateIOVec(data, lenIn);
}

/**
 * Creates an IOVec.
 * param data data to be used.
 * return the IOVec.
 * throws NotImplemented
 */
public static IOVec createIOVec(byte[] data)
    throws BadParameter, NoSuccess, NotImplemented {
    initializeFactory();
    return factory.doCreateIOVec(data);
}

/**
 * Creates an IOVec.
 * param size size of data to be used.
 * param lenIn number of bytes to read/write on readV/writeV.
 * return the IOVec.
 * throws NotImplemented
 */
public static IOVec createIOVec(int size, int lenIn)
    throws BadParameter, NoSuccess, NotImplemented {
    initializeFactory();
    return factory.doCreateIOVec(size, lenIn);
}

/**
```

```
 * Creates an IOVec.
 * param size size of data to be used.
 * return the IOVec.
 * throws NotImplemented
 */
public static IOVec createIOVec(int size)
    throws BadParameter, NoSuccess, NotImplemented {
    initializeFactory();
    return factory.doCreateIOVec(size);
}

/**
 * Creates a File.
 * param session the session handle.
 * param name location of the file.
 * param flags the open mode.
 * return the file instance.
 */
public static File createFile(Session session, URL name, int flags)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, AlreadyExists, DoesNotExist,
        Timeout, NoSuccess {
    initializeFactory();
    return factory.doCreateFile(session, name, flags);
}

/**
 * Creates a FileInputStream.
 * param session the session handle.
 * param name location of the file.
 * return the FileInputStream instance.
 */
public static FileInputStream createFileInputStream(Session session, URL name)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, AlreadyExists, DoesNotExist,
        Timeout, NoSuccess {
    initializeFactory();
    return factory.doCreateFileInputStream(session, name);
}

/**
 * Creates a FileOutputStream.
 * param session the session handle.
 * param name location of the file.
 * param append when set, the file is opened for appending.
 * return the FileOutputStream instance.
 */
public static FileOutputStream createFileOutputStream(Session session, URL name,
```

```
        boolean append)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, AlreadyExists, DoesNotExist,
        Timeout, NoSuccess {
    initializeFactory();
    return factory.doCreateFileOutputStream(session, name, append);
}


/**
 * Creates a FileOutputStream.
 * param session the session handle.
 * param name location of the file.
 * return the FileOutputStream instance.
 */
public static FileOutputStream createFileOutputStream(Session session, URL name)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, AlreadyExists, DoesNotExist,
        Timeout, NoSuccess {
    return createFileOutputStream(session, name, false);
}


/**
 * Creates a File for reading.
 * param session the session handle.
 * param name location of the file.
 * return the file instance.
 */
public static File createFile(Session session, URL name)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, AlreadyExists, DoesNotExist,
        Timeout, NoSuccess {
    initializeFactory();
    return factory.doCreateFile(session, name, Flags.READ.getValue());
}


/**
 * Creates a Directory.
 * param session the session handle.
 * param name location of the directory.
 * param flags the open mode.
 * return the directory instance.
 */
public static Directory createDirectory(Session session, URL name,
        int flags)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, AlreadyExists, DoesNotExist, Timeout, NoSuccess {
```

```
        initializeFactory();
        return factory.doCreateDirectory(session, name, flags);
    }


    /**
     * Creates a Directory for reading.
     * param session the session handle.
     * param name location of the directory.
     * return the directory instance.
     */
    public static Directory createDirectory(Session session, URL name)
        throws NotImplemented, IncorrectURL,
            AuthenticationFailed, AuthorizationFailed, PermissionDenied,
            BadParameter, AlreadyExists, DoesNotExist, Timeout, NoSuccess {
        initializeFactory();
        return factory.doCreateDirectory(session, name, Flags.READ.getValue());
    }


    /**
     * Creates a Task that creates a File.
     * param mode the task mode.
     * param session the session handle.
     * param name location of the file.
     * param flags the open mode.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public static Task<File> createFile(TaskMode mode,
            Session session, URL name, int flags)
        throws NotImplemented {
        initializeFactory();
        return factory.doCreateFile(mode, session, name, flags);
    }


    /**
     * Creates a Task that creates a File for reading.
     * param mode the task mode.
     * param session the session handle.
     * param name location of the file.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public static Task<File> createFile(TaskMode mode,
            Session session, URL name)
        throws NotImplemented {
        initializeFactory();
        return factory.doCreateFile(mode, session, name, Flags.READ.getValue());
    }
```

```
/**
 * Creates a Task that creates a FileInputStream.
 * param mode the task mode.
 * param session the session handle.
 * param name location of the file.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public static Task<FileInputStream> createFileInputStream(TaskMode mode,
        Session session, URL name)
    throws NotImplemented {
    initializeFactory();
    return factory.doCreateFileInputStream(mode, session, name);
}


/**
 * Creates a Task that creates a FileOutputStream.
 * param mode the task mode.
 * param session the session handle.
 * param name location of the file.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public static Task<FileOutputStream> createFileOutputStream(TaskMode mode,
        Session session, URL name)
    throws NotImplemented {
    return createFileOutputStream(mode, session, name, false);
}


/**
 * Creates a Task that creates a FileOutputStream.
 * param mode the task mode.
 * param session the session handle.
 * param name location of the file.
 * param append when set, the file is opened for appending.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public static Task<FileOutputStream> createFileOutputStream(TaskMode mode,
        Session session, URL name, boolean append)
    throws NotImplemented {
    initializeFactory();
    return factory.doCreateFileOutputStream(mode, session, name, append);
}


/**
```

```
 * Creates a Task that creates a Directory.
 * param mode the task mode.
 * param session the session handle.
 * param name location of the directory.
 * param flags the open mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public static Task<Directory> createDirectory(TaskMode mode,
        Session session, URL name, int flags)
    throws NotImplemented {
    initializeFactory();
    return factory.doCreateDirectory(mode, session, name, flags);
}


/**
 * Creates a Task that creates a Directory for reading.
 * param mode the task mode.
 * param session the session handle.
 * param name location of the directory.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public static Task<Directory> createDirectory(TaskMode mode,
        Session session, URL name)
    throws NotImplemented {
    initializeFactory();
    return factory.doCreateDirectory(mode, session, name, Flags.READ.getValue());
}


/**
 * Creates a File using the default session.
 * param name location of the file.
 * param flags the open mode.
 * return the file instance.
 */
public static File createFile(URL name, int flags)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, AlreadyExists, DoesNotExist,
        Timeout, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateFile(session, name, flags);
}


/**
 * Creates a File for reading, using the default session.
```

```
 * param name location of the file.
 * return the file instance.
 */
public static File createFile(URL name)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, AlreadyExists, DoesNotExist,
        Timeout, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateFile(session, name, Flags.READ.getValue());
}


/**
 * Creates a Directory, using the default session.
 * param name location of the directory.
 * param flags the open mode.
 * return the directory instance.
 */
public static Directory createDirectory(URL name, int flags)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, AlreadyExists, DoesNotExist, Timeout, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateDirectory(session, name, flags);
}


/**
 * Creates a Directory for reading, using the default session.
 * param name location of the directory.
 * return the directory instance.
 */
public static Directory createDirectory(URL name)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, AlreadyExists, DoesNotExist, Timeout, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateDirectory(session, name, Flags.READ.getValue());
}


/**
 * Creates a Task that creates a File, using the default session.
 * param mode the task mode.
 * param name location of the file.
 * param flags the open mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
```

```
 * exception NoSuccess is thrown when the default session could not be
 *     created.
 */
public static Task<File> createFile(TaskMode mode, URL name, int flags)
    throws NotImplemented, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateFile(mode, session, name, flags);
}

/**
 * Creates a Task that creates a File for reading, using the default session.
 * param mode the task mode.
 * param name location of the file.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 * exception NoSuccess is thrown when the default session could not be
 *     created.
 */
public static Task<File> createFile(TaskMode mode, URL name)
    throws NotImplemented, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateFile(mode, session, name, Flags.READ.getValue());
}

/**
 * Creates a Task that creates a Directory, using the default session.
 * param mode the task mode.
 * param name location of the directory.
 * param flags the open mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 * exception NoSuccess is thrown when the default session could not be
 *     created.
 */
public static Task<Directory> createDirectory(TaskMode mode,
        URL name, int flags)
    throws NotImplemented, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateDirectory(mode, session, name, flags);
}

/**
 * Creates a Task that creates a Directory for reading, using the default session.
 * param mode the task mode.
 * param name location of the directory.
```

```
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented
     * exception NoSuccess is thrown when the default session could not be
     *     created.
     */
    public static Task<Directory> createDirectory(TaskMode mode, URL name)
        throws NotImplemented, NoSuccess {
        Session session = SessionFactory.createSession();
        initializeFactory();
        return factory.doCreateDirectory(mode, session, name, Flags.READ.getValue());
    }
}
```

4.3.5   FileInputStream

```
package org.ogf.saga.file;

import java.io.InputStream;

import org.ogf.saga.SagaObject;

/**
 * Since Java programmers are used to streams, the Java language bindings of
 * SAGA provide them. In contrast to everything else in the language bindings,
 * this is an abstract class, not an interface, because it is supposed to be
 * a java.io.InputStream (which is a class, not an interface).
 * Implementations should redefine methods of java.io.InputStream.
 * TODO: Should we have tasking versions of these methods???
 */
public abstract class FileInputStream extends InputStream implements SagaObject {

    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}
```

4.3.6   FileOutputStream

```
package org.ogf.saga.file;

import java.io.OutputStream;

import org.ogf.saga.SagaObject;
```

```
/**
 * Since Java programmers are used to streams, the Java language bindings of
 * SAGA provide them. In contrast to everything else in the language bindings,
 * this is an abstract class, not an interface, because it is supposed to be
 * a java.io.OutputStream (which is a class, not an interface).
 * Implementations should redefine methods of java.io.OutputStream.
 * TODO: Should we have tasking versions of these methods???
 */
public abstract class FileOutputStream extends OutputStream implements SagaObject {

    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}
```

4.3.7   SeekMode

```
package org.ogf.saga.file;

/**
 * Determines the seekmode for {link File#seek}:
 * seek from start, current position, or end.
 */
public enum SeekMode {
    START(1),
    CURRENT(2),
    END(3);

    private int value;

    SeekMode(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }
}
```

4.4 SAGA Replica Management

This section of the SAGA API describes the interaction with replica systems.
Numerous SAGA use cases required replica management functionality in the
API – however, only a small number of operation have been requested. The
methods described here are hence limited to the creation and maintainance of
logical files, replicas, and to search on logical file meta data.

4.4.1 LogicalDirectory

```
package org.ogf.saga.logicalfile;

import java.util.List;

import org.ogf.saga.URL;
import org.ogf.saga.attributes.AsyncAttributes;
import org.ogf.saga.error.AlreadyExists;
import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.DoesNotExist;
import org.ogf.saga.error.IncorrectState;
import org.ogf.saga.error.IncorrectURL;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;
import org.ogf.saga.namespace.NSDirectory;
import org.ogf.saga.task.TaskMode;
import org.ogf.saga.task.Task;

/**
 * This interface represents a container for logical files in a logical
 * file name space.
 */
public interface LogicalDirectory extends NSDirectory, AsyncAttributes {

    /**
     * Tests the name for being a logical file.
     * Is an alias for {link NSDirectory#isEntry}.
     * param name to be tested.
     * return <code>true</code> if the name represents a non-directory entry.
     */
    public boolean isFile(URL name)
        throws NotImplemented, IncorrectURL, AuthenticationFailed,
            AuthorizationFailed, PermissionDenied, BadParameter,
            DoesNotExist, IncorrectState, Timeout, NoSuccess;
```

```
/**
 * Finds entries in the current directory and possibly below, with matching names
 * and matching meta data.
 * param namePattern pattern for names of entries to be found.
 * param attrPattern  pattern for meta data keys/values of entries to be
 *          found.
 * param flags       flags defining the operation modus.
 * return the list of matching entries.
 */
public List<URL> find(String namePattern, String[] attrPattern,
        int flags)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, IncorrectState, Timeout,
        NoSuccess;

/**
 * Finds entries in the current directory and below, with matching names
 * and matching meta data.
 * param namePattern pattern for names of entries to be found.
 * param attrPattern  pattern for meta data keys/values of entries to be
 *          found.
 * return the list of matching entries.
 */
public List<URL> find(String namePattern, String[] attrPattern)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
        PermissionDenied, BadParameter, IncorrectState, Timeout,
        NoSuccess;

// openLogicalDir and openLogicalFile: names changed with respect
// to specs  because of Java restriction: cannot redefine methods with
// just a different return type.

/**
 * Creates a new <code>LogicalDirectory</code> instance.
 * param name directory to open.
 * param flags defining the operation modus.
 * return the opened directory instance.
 */
public LogicalDirectory openLogicalDir(URL name, int flags)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, IncorrectState, AlreadyExists, DoesNotExist,
        Timeout, NoSuccess;

/**
 * Creates a new <code>LogicalDirectory</code> instance with read flag.
 * param name directory to open.
 * return the opened directory instance.
 */
```

```
public LogicalDirectory openLogicalDir(URL name)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, IncorrectState, AlreadyExists, DoesNotExist,
        Timeout, NoSuccess;

/**
 * Creates a new <code>LogicalFile</code> instance.
 * param name logical file to open.
 * param flags defining the operation modus.
 * return the opened logical file.
 */
public LogicalFile openLogicalFile(URL name, int flags)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, IncorrectState, AlreadyExists, DoesNotExist,
        Timeout, NoSuccess;

/**
 * Creates a new <code>LogicalFile</code> instance with read flag.
 * param name logical file to open.
 * return the opened logical file.
 */
public LogicalFile openLogicalFile(URL name)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, IncorrectState, AlreadyExists, DoesNotExist,
        Timeout, NoSuccess;

// Task versions ...

/**
 * Creates a task that tests the name for being a logical file.
 * Is an alias for {link NSDirectory#isEntry}.
 * param mode the task mode.
 * param name to be tested.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<Boolean> isFile(TaskMode mode, URL name)
    throws NotImplemented ;

/**
 * Creates a task that finds entries in the current directory and below,
 * with matching names and matching meta data.
 * param mode        the task mode.
 * param namePattern pattern for names of entries to be found.
 * param attrPattern pattern for meta data keys/values of entries to be
 *            found.
```

```
 * param flags       flags defining the operation modus.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<List<URL>> find(TaskMode mode, String namePattern,
        String[] attrPattern, int flags)
    throws NotImplemented;


/**
 * Creates a task that finds entries in the current directory and below,
 * with matching names and matching meta data.
 * param mode        the task mode.
 * param namePattern pattern for names of entries to be found.
 * param attrPattern pattern for meta data keys/values of entries to be
 *           found.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<List<URL>> find(TaskMode mode, String namePattern,
        String[] attrPattern)
    throws NotImplemented;


/**
 * Creates a task that creates a new <code>LogicalDirectory</code>
 * instance.
 * param mode the task mode.
 * param name directory to open.
 * param flags defining the operation modus.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<LogicalDirectory> openLogicalDir(TaskMode mode, URL name,
        int flags)
    throws NotImplemented;


/**
 * Creates a task that creates a new <code>LogicalDirectory</code>
 * instance.
 * param mode the task mode.
 * param name directory to open.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<LogicalDirectory> openLogicalDir(TaskMode mode, URL name)
    throws NotImplemented;
```

```
    /**
     * Creates a task that creates a new <code>LogicalFile</code> instance.
     * param mode the task mode.
     * param name the file to open.
     * param flags defining the operation modus.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *      method is not implemented.
     */
    public Task<LogicalFile> openLogicalFile(TaskMode mode, URL name,
              int flags)
        throws NotImplemented;

    /**
     * Creates a task that creates a new <code>LogicalFile</code> instance.
     * param mode the task mode.
     * param name the file to open.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *      method is not implemented.
     */
    public Task<LogicalFile> openLogicalFile(TaskMode mode, URL name)
        throws NotImplemented;
}
```

4.4.2   LogicalFile

```
package org.ogf.saga.logicalfile;

import java.util.List;

import org.ogf.saga.URL;
import org.ogf.saga.attributes.AsyncAttributes;
import org.ogf.saga.error.AlreadyExists;
import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.DoesNotExist;
import org.ogf.saga.error.IncorrectState;
import org.ogf.saga.error.IncorrectURL;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;
import org.ogf.saga.namespace.NSEntry;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;
```

```
/**
 * A LogicalFile provides the means to handle the contents of logical
 * files.
 */
public interface LogicalFile extends NSEntry, AsyncAttributes {

    /**
     * Adds a replica location to the replica set.
     * Note: does never throw an <code>AlreadyExists</code> exception!
     * param name the location to add.
     */
    public void addLocation(URL name)
        throws NotImplemented, IncorrectURL, AuthenticationFailed,
            AuthorizationFailed, PermissionDenied, BadParameter,
            IncorrectState, Timeout, NoSuccess;

    /**
     * Removes a replica location from the replica set.
     * param name the location to remove.
     */
    public void removeLocation(URL name)
        throws NotImplemented, IncorrectURL, AuthenticationFailed,
            AuthorizationFailed, PermissionDenied, BadParameter,
            IncorrectState, DoesNotExist, Timeout, NoSuccess;

    /**
     * Changes a replica location in the replica set.
     * param nameOld the location to be updated.
     * param nameNew the updated location.
     */
    public void updateLocation(URL nameOld, URL nameNew)
        throws NotImplemented, IncorrectURL, AuthenticationFailed,
            AuthorizationFailed, PermissionDenied, BadParameter,
            IncorrectState, AlreadyExists, DoesNotExist, Timeout, NoSuccess;

    /**
     * Lists the locations in this location set.
     * return the location list.
     */
    public List<URL> listLocations()
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, IncorrectState, Timeout, NoSuccess;

    /**
     * Replicates a file from any of the known locations to a new location.
     * param name location to replicate to.
     * param flags flags defining the operation modus.
     */
    public void replicate(URL name, int flags)
        throws NotImplemented, IncorrectURL, AuthenticationFailed,
```

```
            AuthorizationFailed, PermissionDenied, BadParameter,
            IncorrectState, AlreadyExists, DoesNotExist, Timeout, NoSuccess;

  /**
   * Replicates a file from any of the known locations to a new location,
   * with default flags NONE.
   * param name location to replicate to.
   */
  public void replicate(URL name)
      throws NotImplemented, IncorrectURL, AuthenticationFailed,
            AuthorizationFailed, PermissionDenied, BadParameter,
            IncorrectState, AlreadyExists, DoesNotExist, Timeout, NoSuccess;

  //
  // Task versions ...
  //

  /**
   * Creates a task that adds a replica location to the replica set.
   * param mode the task mode.
   * param name the location to add.
   * return the task.
   * exception NotImplemented is thrown when the task version of this
   *      method is not implemented.
   */
  public Task addLocation(TaskMode mode, URL name)
      throws NotImplemented;

  /**
   * Creates a task that removes a replica location from the replica set.
   * param mode the task mode.
   * param name the location to remove.
   * return the task.
   * exception NotImplemented is thrown when the task version of this
   *      method is not implemented.
   */
  public Task removeLocation(TaskMode mode, URL name)
      throws NotImplemented;

  /**
   * Creates a task that changes a replica location in the replica set.
   * param mode the task mode.
   * param nameOld the location to be updated.
   * param nameNew the updated location.
   * return the task.
   * exception NotImplemented is thrown when the task version of this
   *      method is not implemented.
   */
  public Task updateLocation(TaskMode mode, URL nameOld, URL nameNew)
      throws NotImplemented;
```

```
    /**
     * Creates a task that lists the locations in this location set.
     * param mode the task mode.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task<List<URL>> listLocations(TaskMode mode)
        throws NotImplemented;

    /**
     * Creates a task that replicates a file from any of the known locations
     * to a new location.
     * param mode the task mode.
     * param name location to replicate to.
     * param flags flags defining the operation modus.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task replicate(TaskMode mode, URL name, int flags)
        throws NotImplemented;

    /**
     * Creates a task that replicates a file from any of the known locations
     * to a new location, with default flags NONE.
     * param mode the task mode.
     * param name location to replicate to.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task replicate(TaskMode mode, URL name)
        throws NotImplemented;
}
```

4.4.3 LogicalFileFactory

```
package org.ogf.saga.logicalfile;

import org.ogf.saga.URL;
import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.AlreadyExists;
import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.DoesNotExist;
```

```
import org.ogf.saga.error.IncorrectURL;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;
import org.ogf.saga.namespace.Flags;
import org.ogf.saga.session.Session;
import org.ogf.saga.session.SessionFactory;
import org.ogf.saga.task.TaskMode;
import org.ogf.saga.task.Task;

/**
 * Factory for objects from the logicalfile package.
 */
public abstract class LogicalFileFactory {

    private static LogicalFileFactory factory;

    private static synchronized void initializeFactory()
        throws NotImplemented {
        if (factory == null) {
            factory = ImplementationBootstrapLoader.createLogicalFileFactory();
        }
    }

    /**
     * Creates a LogicalFile. To be provided by the implementation.
     * param session the session handle.
     * param name location of the file.
     * param flags the open mode.
     * return the file instance.
     */
    protected abstract LogicalFile doCreateLogicalFile(Session session,
            URL name, int flags)
        throws NotImplemented, IncorrectURL,
            AuthenticationFailed, AuthorizationFailed, PermissionDenied,
            BadParameter, AlreadyExists, DoesNotExist,
            Timeout, NoSuccess;

    /**
     * Creates a Directory. To be provided by the implementation.
     * param session the session handle.
     * param name location of directory.
     * param flags the open mode.
     * return the directory instance.
     */
    protected abstract LogicalDirectory doCreateLogicalDirectory(
            Session session, URL name, int flags)
        throws NotImplemented, IncorrectURL,
            AuthenticationFailed, AuthorizationFailed, PermissionDenied,
```

```
            BadParameter, AlreadyExists, DoesNotExist, Timeout, NoSuccess;

    /**
     * Creates a Task that creates a LogicalFile.
     * To be provided by the implementation.
     * param mode the task mode.
     * param session the session handle.
     * param name location of the file.
     * param flags the open mode.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    protected abstract Task<LogicalFile> doCreateLogicalFile(
            TaskMode mode, Session session, URL name, int flags)
        throws NotImplemented;

    /**
     * Creates a Task that creates a LogicalDirectory.
     * To be provided by the implementation.
     * param mode the task mode.
     * param session the session handle.
     * param name location of directory.
     * param flags the open mode.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    protected abstract Task<LogicalDirectory>
            doCreateLogicalDirectory(
                    TaskMode mode, Session session, URL name, int flags)
            throws NotImplemented;

    /**
     * Creates a LogicalFile.
     * param session the session handle.
     * param name location of the file.
     * param flags the open mode.
     * return the file instance.
     */
    public static LogicalFile createLogicalFile(Session session, URL name,
            int flags)
        throws NotImplemented, IncorrectURL,
            AuthenticationFailed, AuthorizationFailed, PermissionDenied,
            BadParameter, AlreadyExists, DoesNotExist,
            Timeout, NoSuccess {
        initializeFactory();
        return factory.doCreateLogicalFile(session, name, flags);
    }
```

```
    /**
     * Creates a LogicalDirectory.
     * param session the session handle.
     * param name location of the directory.
     * param flags the open mode.
     * return the directory instance.
     */
    public static LogicalDirectory createLogicalDirectory(Session session,
            URL name, int flags)
        throws NotImplemented, IncorrectURL,
            AuthenticationFailed, AuthorizationFailed, PermissionDenied,
            BadParameter, AlreadyExists, DoesNotExist, Timeout, NoSuccess {
        initializeFactory();
        return factory.doCreateLogicalDirectory(session, name, flags);
    }


    /**
     * Creates a Task that creates a LogicalFile.
     * param mode the task mode.
     * param session the session handle.
     * param name location of the file.
     * param flags the open mode.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public static Task<LogicalFile> createLogicalFile(
            TaskMode mode, Session session, URL name, int flags)
        throws NotImplemented {
        initializeFactory();
        return factory.doCreateLogicalFile(mode, session, name, flags);
    }


    /**
     * Creates a Task that creates a LogicalDirectory.
     * param mode the task mode.
     * param session the session handle.
     * param name location of the directory.
     * param flags the open mode.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public static Task<LogicalDirectory> createLogicalDirectory(
            TaskMode mode, Session session, URL name, int flags)
        throws NotImplemented {
        initializeFactory();
        return factory.doCreateLogicalDirectory(mode, session, name, flags);
    }
```

```
/**
 * Creates a LogicalFile using READ open mode.
 * param session the session handle.
 * param name location of the file.
 * return the file instance.
 */
public static LogicalFile createLogicalFile(Session session, URL name)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, AlreadyExists, DoesNotExist,
        Timeout, NoSuccess {
    initializeFactory();
    return factory.doCreateLogicalFile(session, name, Flags.READ.getValue());
}


/**
 * Creates a LogicalDirectory using READ open mode.
 * param session the session handle.
 * param name location of the directory.
 * return the directory instance.
 */
public static LogicalDirectory createLogicalDirectory(Session session,
        URL name)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, AlreadyExists, DoesNotExist, Timeout, NoSuccess {
    initializeFactory();
    return factory.doCreateLogicalDirectory(session, name, Flags.READ.getValue());
}


/**
 * Creates a Task that creates a LogicalFile using READ open mode.
 * param mode the task mode.
 * param session the session handle.
 * param name location of the file.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public static Task<LogicalFile> createLogicalFile(
        TaskMode mode, Session session, URL name)
    throws NotImplemented {
    initializeFactory();
    return factory.doCreateLogicalFile(mode, session, name, Flags.READ.getValue());
}


/**
 * Creates a Task that creates a LogicalDirectory using READ open mode.
 * param mode the task mode.
 * param session the session handle.
```

```
 * param name location of the directory.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *      method is not implemented.
 */
public static Task<LogicalDirectory> createLogicalDirectory(
        TaskMode mode, Session session, URL name)
    throws NotImplemented {
    initializeFactory();
    return factory.doCreateLogicalDirectory(mode, session, name, Flags.READ.getValue());
}


/**
 * Creates a LogicalFile using the default session.
 * param name location of the file.
 * param flags the open mode.
 * return the file instance.
 */
public static LogicalFile createLogicalFile(URL name,int flags)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, AlreadyExists, DoesNotExist,
        Timeout, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateLogicalFile(session, name, flags);
}


/**
 * Creates a LogicalDirectory using the default session.
 * param name location of the directory.
 * param flags the open mode.
 * return the directory instance.
 */
public static LogicalDirectory createLogicalDirectory(URL name, int flags)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, AlreadyExists, DoesNotExist, Timeout, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateLogicalDirectory(session, name, flags);
}


/**
 * Creates a Task that creates a LogicalFile using the default session.
 * param mode the task mode.
 * param name location of the file.
 * param flags the open mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
```

```
   *      method is not implemented.
   * exception NoSuccess is thrown when the default session could not be
   *      created.
   */
  public static Task<LogicalFile> createLogicalFile(
          TaskMode mode, URL name, int flags)
      throws NotImplemented, NoSuccess {
      Session session = SessionFactory.createSession();
      initializeFactory();
      return factory.doCreateLogicalFile(mode, session, name, flags);
  }


  /**
   * Creates a Task that creates a LogicalDirectory using the default session.
   * param mode the task mode.
   * param name location of the directory.
   * param flags the open mode.
   * return the task.
   * exception NotImplemented is thrown when the task version of this
   *      method is not implemented.
   * exception NoSuccess is thrown when the default session could not be
   *      created.
   */
  public static Task<LogicalDirectory> createLogicalDirectory(
          TaskMode mode, URL name, int flags)
      throws NotImplemented, NoSuccess {
      Session session = SessionFactory.createSession();
      initializeFactory();
      return factory.doCreateLogicalDirectory(mode, session, name, flags);
  }


  /**
   * Creates a LogicalFile using READ open mode, using the default session .
   * param name location of the file.
   * return the file instance.
   */
  public static LogicalFile createLogicalFile(URL name)
      throws NotImplemented, IncorrectURL,
          AuthenticationFailed, AuthorizationFailed, PermissionDenied,
          BadParameter, AlreadyExists, DoesNotExist,
          Timeout, NoSuccess {
      Session session = SessionFactory.createSession();
      initializeFactory();
      return factory.doCreateLogicalFile(session, name, Flags.READ.getValue());
  }


  /**
   * Creates a LogicalDirectory using READ open mode, using the default session.
   * param name location of the directory.
   * return the directory instance.
```

```
      */
    public static LogicalDirectory createLogicalDirectory(URL name)
        throws NotImplemented, IncorrectURL,
            AuthenticationFailed, AuthorizationFailed, PermissionDenied,
            BadParameter, AlreadyExists, DoesNotExist, Timeout, NoSuccess {
        Session session = SessionFactory.createSession();
        initializeFactory();
        return factory.doCreateLogicalDirectory(session, name, Flags.READ.getValue());
    }

    /**
     * Creates a Task that creates a LogicalFile using READ open mode,
     * using the default session.
     * param mode the task mode.
     * param name location of the file.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     * exception NoSuccess is thrown when the default session could not be
     *     created.
     */
    public static Task<LogicalFile> createLogicalFile(TaskMode mode, URL name)
        throws NotImplemented, NoSuccess {
        Session session = SessionFactory.createSession();
        initializeFactory();
        return factory.doCreateLogicalFile(mode, session, name, Flags.READ.getValue());
    }

    /**
     * Creates a Task that creates a LogicalDirectory using READ open mode,
     * using the default session.
     * param mode the task mode.
     * param name location of the directory.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     * exception NoSuccess is thrown when the default session could not be
     *     created.
     */
    public static Task<LogicalDirectory> createLogicalDirectory(TaskMode mode, URL name)
        throws NotImplemented, NoSuccess {
        Session session = SessionFactory.createSession();
        initializeFactory();
        return factory.doCreateLogicalDirectory(mode, session, name, Flags.READ.getValue());
    }
}
```

4.5 SAGA Streams

A number of use cases involve launching of remotely located components in order to create distributed applications. These use cases require simple remote socket connections to be established between these components and their control interfaces.

The target of the streams API is to establish the simplest possible authenticated socket connection with hooks to support application level authorization.

In the Java language bindings, the `Activity` type is an enumeration, with a method to create an integer value from it, methods to create bit masks, and a method to examine if a specific `Activity` is set in a bit mask.

In the `Stream` interface, the `wait` method is renamed to `waitStream`, because it is not supposed to redefine the `java.lang.Object wait` method. Also, as mentioned earlier, for Java, it is not appropriate to return POSIX error codes for the `read` and `write` methods. Instead, they throw a `java.io.IOException` when an error occurs. This is less informative than a specific error code, but is more "Java-like", and probably better implementable on existing Java Grid middleware.

4.5.1 Stream

```
package org.ogf.saga.stream;

import java.io.IOException;

import org.ogf.saga.SagaObject;
import org.ogf.saga.URL;
import org.ogf.saga.attributes.AsyncAttributes;
import org.ogf.saga.buffer.Buffer;
import org.ogf.saga.context.Context;
import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.IncorrectState;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;
import org.ogf.saga.monitoring.AsyncMonitorable;
import org.ogf.saga.task.Async;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;
```

```
/**
 * A client stream object.
 */
public interface Stream extends SagaObject, Async, AsyncAttributes,
       AsyncMonitorable {

    // Optional attributes

    /** Attribute name, determines size of send buffer. */
    public static final String BUFSIZE = "Bufsize";

    /**
     * Attribute name, determines the amount of idle time before dropping
     * the connection, in seconds.
     */
    public static final String TIMEOUT = "Timeout";

    /**
     * Attribute name, determines if read/writes are blocking or not.
     * If this attribute is not supported, implementation must do blocking.
     */
    public static final String BLOCKING = "Blocking";

    /**
     * Attribute name, determines if data are compressed before/after transfer.
     */
    public static final String COMPRESSION = "Compression";

    /** Attribute name, determines if packets are sent without delay. */
    public static final String NODELAY = "Nodelay";

    /** Attribute name, determines if all sent data MUST arrive. */
    public static final String RELIABLE = "Reliable";

    // Metrics

    /**
     * Metric name, fires if the state of the stream changes, and has the
     * value of the new state.
     */
    public static final String STREAM_STATE = "stream.state";

    /**
     * Metric name, fires if the stream gets readable (which means that a
     * subsequent read() can successfully read one or more bytes of data).
     */
    public static final String STREAM_READ = "stream.read";

    /**
     * Metric name, fires if the stream gets writable (which means that a
```

```
 * subsequent write() can successfully write one or more bytes of data).
 */
public static final String STREAM_WRITE = "stream.write";

/** Metric name, fires if the stream has an error condition. */
public static final String STREAM_EXCEPTION = "stream.exception";

/** Metric name, fires if the stream gets dropped by the remote party. */
public static final String STREAM_DROPPED = "stream.dropped";

// inspection methods

/**
 * Obtains the URL that was used to create the stream.
 * When this stream is the result of a {link StreamService#serve()}
 * call, <code>null</code> is returned.
 * return the URL.
 */
public URL getUrl()
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
           PermissionDenied, IncorrectState, Timeout, NoSuccess;

/**
 * Returns the remote authorization info.
 * The returned context is deep-copied.
 * return the remote context.
 */
public Context getContext()
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
           PermissionDenied, IncorrectState, Timeout, NoSuccess;

// management methods

/**
 * Establishes a connection to the target defined during the construction
 * of the stream.
 */
public void connect()
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
           PermissionDenied, IncorrectState, Timeout, NoSuccess;

/**
 * Checks if the stream is ready for I/O, or if it has entered the
 * ERROR state. It will only check for the specified activities.
 * This method blocks until one or more of the specified activities
 * apply.
 * param what the activities to wait for.
 * return the activities that apply.
 */
public int waitStream(int what)
```

```
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, IncorrectState, NoSuccess;


  /**
   * Checks if the stream is ready for I/O, or if it has entered the
   * ERROR state. It will only check for the specified activities.
   * If the timeout expires, an empty list is returned.
   * param what the activities to wait for.
   * param timeoutInSeconds the timeout in seconds.
   * return the activities that apply.
   */
  public int waitStream(int what, float timeoutInSeconds)
      throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, IncorrectState, NoSuccess;


  /**
   * Closes an active connection.
   * This method performs a non-blocking close.
   * I/O is no longer possible. The stream is put in state CLOSED.
   */
  public void close()
      throws NotImplemented, IncorrectState, NoSuccess;


  /**
   * Closes an active connection.
   * I/O is no longer possible. The stream is put in state CLOSED.
   * param timeoutInSeconds the timeout in seconds.
   */
  public void close(float timeoutInSeconds)
      throws NotImplemented, IncorrectState, NoSuccess;


  // I/O methods


  /**
   * Reads a raw buffer from the stream.
   * param len the maximum number of bytes to be read.
   * param buffer the buffer to store into.
   * return the number of bytes read.
   * exception IOException deviation from the SAGA specs: thrown in case
   *     of an error, where the SAGA specs describe a return of a negative
   *     value, corresponding to negatives of the respective ERRNO error
   *     code.
   */
  public int read(Buffer buffer, int len)
      throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
            PermissionDenied, BadParameter, IncorrectState, Timeout,
            NoSuccess, IOException;


  /**
   * Reads a raw buffer from the stream.
```

```
 * param buffer the buffer to store into.
 * return the number of bytes read.
 * exception IOException deviation from the SAGA specs: thrown in case
 *     of an error, where the SAGA specs describe a return of a negative
 *     value, corresponding to negatives of the respective ERRNO error
 *     code.
 */
public int read(Buffer buffer)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
           PermissionDenied, BadParameter, IncorrectState, Timeout,
           NoSuccess, IOException;


/**
 * Writes a raw buffer to the stream.
 * Note: if the buffer contains less data than the specified len, only
 * the data in the buffer are written.
 * param len the number of bytes of data in the buffer.
 * param buffer the data to be sent.
 * return the number of bytes written.
 * exception IOException deviation from the SAGA specs: thrown in case
 *     of an error, where the SAGA specs describe a return of a negative
 *     value, corresponding to negatives of the respective ERRNO error
 *     code.
 */
public int write(Buffer buffer, int len)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
           PermissionDenied, BadParameter, IncorrectState, Timeout,
           NoSuccess, IOException;


/**
 * Writes a raw buffer to the stream.
 * param buffer the data to be sent.
 * return the number of bytes written.
 * exception IOException deviation from the SAGA specs: thrown in case
 *     of an error, where the SAGA specs describe a return of a negative
 *     value, corresponding to negatives of the respective ERRNO error
 *     code.
 */
public int write(Buffer buffer)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
           PermissionDenied, BadParameter, IncorrectState, Timeout,
           NoSuccess, IOException;
//
// Task versions ...
//

// inspection methods

/**
 * Creates a task that obtains the URL that was used to create the stream.
```

```
 * When this stream is the result of a {link StreamService#serve()}
 * call, the URL will be <code>null</code>.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<URL> getUrl(TaskMode mode)
    throws NotImplemented;

/**
 * Creates a task that obtains the remote authorization info.
 * The returned context is deep-copied.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<Context> getContext(TaskMode mode)
    throws NotImplemented;

// management methods

/**
 * Returns a task that
 * establishes a connection to the target defined during the construction
 * of the stream.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task connect(TaskMode mode)
    throws NotImplemented;

/**
 * Returns a task that
 * checks if the stream is ready for I/O, or if it has entered the
 * ERROR state. It will only check for the specified activities.
 * param mode the task mode.
 * param what the activities to wait for.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<Integer> waitStream(TaskMode mode, int what)
    throws NotImplemented;

/**
 * Returns a task that
```

```
    * checks if the stream is ready for I/O, or if it has entered the
    * ERROR state. It will only check for the specified activities.
    * If the timeout expires, the task will return an empty list.
    * param mode the task mode.
    * param what the activities to wait for.
    * param timeoutInSeconds the timout in seconds.
    * return the task.
    * exception NotImplemented is thrown when the task version of this
    *     method is not implemented.
    */
  public Task<Integer> waitStream(TaskMode mode, int what,
          float timeoutInSeconds)
      throws NotImplemented;


  /**
   * Returns a task that closes an active connection.
   * param mode the task mode.
   * return the task.
   * exception NotImplemented is thrown when the task version of this
   *     method is not implemented.
   */
  public Task close(TaskMode mode)
      throws NotImplemented;


  /**
   * Returns a task that closes an active connection.
   * param mode the task mode.
   * param timeoutInSeconds the timeout in seconds.
   * return the task.
   * exception NotImplemented is thrown when the task version of this
   *     method is not implemented.
   */
  public Task close(TaskMode mode, float timeoutInSeconds)
      throws NotImplemented;


  // I/O methods

  /**
   * Creates a task that reads a raw buffer from the stream.
   * param mode the task mode.
   * param len the maximum number of bytes to be read.
   * param buffer the buffer to store into.
   * return the task.
   * exception NotImplemented is thrown when the task version of this
   *     method is not implemented.
   */
  public Task<Integer> read(TaskMode mode, Buffer buffer, int len)
      throws NotImplemented;


  /**
```

```
     * Creates a task that reads a raw buffer from the stream.
     * param mode the task mode.
     * param buffer the buffer to store into.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task<Integer> read(TaskMode mode, Buffer buffer)
        throws NotImplemented;

    /**
     * Creates a task that writes a raw buffer to the stream.
     * Note: if the buffer contains less data than the specified len, only
     * the data in the buffer are written.
     * param mode the task mode.
     * param len the number of bytes of data in the buffer.
     * param buffer the data to be sent.
     * return the number of bytes written.
     * exception IOException deviation from the SAGA specs: thrown in case
     *     of an error.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task<Integer> write(TaskMode mode, Buffer buffer, int len)
        throws NotImplemented;

    /**
     * Creates a task that writes a raw buffer to the stream.
     * param mode the task mode.
     * param buffer the data to be sent.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public Task<Integer> write(TaskMode mode, Buffer buffer)
        throws NotImplemented;
}
```

4.5.2 StreamService

```
package org.ogf.saga.stream;

import org.ogf.saga.SagaObject;
import org.ogf.saga.URL;
import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.IncorrectState;
```

```
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;
import org.ogf.saga.monitoring.AsyncMonitorable;
import org.ogf.saga.permissions.Permissions;
import org.ogf.saga.task.Async;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;


/**
 * A <code>StreamService</code> object establishes a listening/server object
 * that waits for client connections. It is similar to a serversocket.
 */
public interface StreamService extends SagaObject, Async, AsyncMonitorable,
        Permissions {

    // Metrics

    /** Metric name, fires if a client connects. */
    public static final String STREAMSERVER_CLIENTCONNECT
            = "stream_server.client_connect";

    // Methods

    /**
     * Obtains the URL to be used to connect to this server.
     * return the URL.
     */
    public URL getUrl()
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
                PermissionDenied, IncorrectState, Timeout, NoSuccess;

    /**
     * Waits for incoming client connections (like an accept of a
     * serversocket). The returned stream is in OPEN state.
     * This call may block indefinately.
     * return the client connection.
     */
    public Stream serve()
        throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
                PermissionDenied, IncorrectState, Timeout, NoSuccess;

    /**
     * Waits for incoming client connections (like an accept of a
     * serversocket). The returned stream is in OPEN state.
     * param timeoutInSeconds the timeout in seconds.
     * return the client connection, or <code>null</code> if the timeout
     * expires before a client connects.
     */
```

```
public Stream serve(float timeoutInSeconds)
    throws NotImplemented, AuthenticationFailed, AuthorizationFailed,
           PermissionDenied, IncorrectState, Timeout, NoSuccess;

/**
 * Closes a stream service.
 * This is a non-blocking call.
 */
public void close()
    throws NotImplemented, IncorrectState, NoSuccess;

/**
 * Closes a stream service.
 * param timeoutInSeconds the timeout in seconds.
 */
public void close(float timeoutInSeconds)
    throws NotImplemented, IncorrectState, NoSuccess;

//
// Task versions ...
//

/**
 * Obtains a task to obtain the URL to be used to connect to this server.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<URL> getUrl(TaskMode mode)
    throws NotImplemented;

/**
 * Obtains a task that waits for incoming client connections
 * (like an accept of a serversocket).
 * The returned stream is in OPEN state.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<Stream> serve(TaskMode mode)
    throws NotImplemented;

/**
 * Obtains a task that waits for incoming client connections
 * (like an accept of a serversocket).
 * The returned stream is in OPEN state.
 * param mode the task mode.
 * param timeoutInSeconds the timeout in seconds.
```

```
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task<Stream> serve(TaskMode mode, float timeoutInSeconds)
    throws NotImplemented;

/**
 * Obtains a task that closes a stream service.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task close(TaskMode mode)
    throws NotImplemented;

/**
 * Obtains a task that closes a stream service.
 * param mode the task mode.
 * param timeoutInSeconds the timeout in seconds.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task close(TaskMode mode, float timeoutInSeconds)
    throws NotImplemented;
}
```

4.5.3   StreamFactory

```
package org.ogf.saga.stream;

import org.ogf.saga.URL;
import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.IncorrectURL;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;
import org.ogf.saga.session.Session;
import org.ogf.saga.session.SessionFactory;
import org.ogf.saga.task.TaskMode;
import org.ogf.saga.task.Task;
```

```
/**
 * Factory for objects from the stream package.
 */
public abstract class StreamFactory {

    private static StreamFactory factory;

    private static synchronized void initializeFactory()
        throws NotImplemented {
        if (factory == null) {
            factory = ImplementationBootstrapLoader.createStreamFactory();
        }
    }

    /**
     * Creates a Stream. To be provided by the implementation.
     * param session the session handle.
     * param name location of the stream service.
     * return the stream.
     */
    protected abstract Stream doCreateStream(Session session, URL name)
        throws NotImplemented, IncorrectURL, BadParameter,
                AuthenticationFailed, AuthorizationFailed, PermissionDenied,
                Timeout, NoSuccess;

    /**
     * Creates a StreamService. To be provided by the implementation.
     * param session the session handle.
     * param name location of the service.
     * return the service.
     */
    protected abstract StreamService doCreateStreamService(Session session,
            URL name)
        throws NotImplemented, IncorrectURL, BadParameter,
                AuthenticationFailed, AuthorizationFailed, PermissionDenied,
                Timeout, NoSuccess;

    /**
     * Creates a StreamService. To be provided by the implementation.
     * param session the session handle.
     * return the service.
     */
    protected abstract StreamService doCreateStreamService(Session session)
        throws NotImplemented, IncorrectURL, BadParameter,
                AuthenticationFailed, AuthorizationFailed, PermissionDenied,
                Timeout, NoSuccess;

    /**
     * Creates a task that creates a Stream.
     * To be provided by the implementation.
```

```
 * param mode the task mode.
 * param session the session handle.
 * param name location of the stream service.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
protected abstract Task<Stream> doCreateStream(TaskMode mode,
        Session session, URL name)
    throws NotImplemented;

/**
 * Creates a task that creates a StreamService.
 * To be provided by the implementation.
 * param mode the task mode.
 * param session the session handle.
 * param name location of the service.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
protected abstract Task<StreamService> doCreateStreamService(
        TaskMode mode, Session session, URL name)
    throws NotImplemented;

/**
 * Creates a task that creates a StreamService.
 * To be provided by the implementation.
 * param mode the task mode.
 * param session the session handle.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
protected abstract Task<StreamService> doCreateStreamService(
        TaskMode mode, Session session)
    throws NotImplemented;

/**
 * Creates a Stream.
 * param session the session handle.
 * param name location of the stream service.
 * return the stream.
 */
public static Stream createStream(Session session, URL name)
    throws NotImplemented, IncorrectURL, BadParameter,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        Timeout, NoSuccess {
    initializeFactory();
    return factory.doCreateStream(session, name);
```

```
    }

    /**
     * Creates a StreamService.
     * param session the session handle.
     * param name location of the service.
     * return the service.
     */
    public static StreamService createStreamService(Session session, URL name)
        throws NotImplemented, IncorrectURL, BadParameter,
            AuthenticationFailed, AuthorizationFailed, PermissionDenied,
            Timeout, NoSuccess {
        initializeFactory();
        return factory.doCreateStreamService(session, name);
    }

    /**
     * Creates a StreamService.
     * param session the session handle.
     * return the service.
     */
    public static StreamService createStreamService(Session session)
        throws NotImplemented, IncorrectURL, BadParameter,
            AuthenticationFailed, AuthorizationFailed, PermissionDenied,
            Timeout, NoSuccess {
        initializeFactory();
        return factory.doCreateStreamService(session);
    }

    /**
     * Creates a Task that creates a Stream.
     * param mode the task mode.
     * param session the session handle.
     * param name location of the stream service.
     * return the task.
     * exception NotImplemented is thrown when the task version of this
     *     method is not implemented.
     */
    public static Task<Stream> createStream(TaskMode mode, Session session,
            URL name) throws NotImplemented {
        initializeFactory();
        return factory.doCreateStream(mode, session, name);
    }

    /**
     * Creates a Task that creates a StreamService.
     * param mode the task mode.
     * param session the session handle.
     * param name location of the service.
     * return the task.
```

```
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public static Task<StreamService> createStreamService(TaskMode mode,
        Session session, URL name) throws NotImplemented {
    initializeFactory();
    return factory.doCreateStreamService(mode, session, name);
}


/**
 * Creates a Task that creates a StreamService.
 * param mode the task mode.
 * param session the session handle.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public static Task<StreamService> createStreamService(TaskMode mode,
        Session session) throws NotImplemented {
    initializeFactory();
    return factory.doCreateStreamService(mode, session);
}


/**
 * Creates a Stream using the default session.
 * param name location of the stream service.
 * return the stream.
 */
public static Stream createStream(URL name)
    throws NotImplemented, IncorrectURL, BadParameter,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        Timeout, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateStream(session, name);
}


/**
 * Creates a StreamService using the default session.
 * param name location of the service.
 * return the service.
 */
public static StreamService createStreamService(URL name)
    throws NotImplemented, IncorrectURL, BadParameter,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        Timeout, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateStreamService(session, name);
}
```

```
/**
 * Creates a StreamService using the default session.
 * return the service.
 */
public static StreamService createStreamService()
    throws NotImplemented, IncorrectURL, BadParameter,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        Timeout, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateStreamService(session);
}


/**
 * Creates a Task that creates a Stream using the default session.
 * param mode the task mode.
 * param name location of the stream service.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 * exception NoSuccess is thrown when the default session could not be
 *     created.
 */
public static Task<Stream> createStream(TaskMode mode, URL name)
    throws NotImplemented, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateStream(mode, session, name);
}


/**
 * Creates a Task that creates a StreamService using the default session.
 * param mode the task mode.
 * param name location of the service.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 * exception NoSuccess is thrown when the default session could not be
 *     created.
 */
public static Task<StreamService> createStreamService(TaskMode mode,
        URL name) throws NotImplemented, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateStreamService(mode, session, name);
}


/**
 * Creates a Task that creates a StreamService using the default session.
```

```
    * param mode the task mode.
    * return the task.
    * exception NotImplemented is thrown when the task version of this
    *     method is not implemented.
    * exception NoSuccess is thrown when the default session could not be
    *     created.
    */
   public static Task<StreamService> createStreamService(TaskMode mode)
           throws NotImplemented, NoSuccess {
       Session session = SessionFactory.createSession();
       initializeFactory();
       return factory.doCreateStreamService(mode, session);
   }
}
```

4.5.4 Activity

```
package org.ogf.saga.stream;

/**
 * Flags for activities of a stream. An application can poll for these
 * events or get asynchronous notification of these events by using
 * metrics.
 * These flags are meant to be or-ed together, resulting in an integer,
 * so methods are added to test for presence and or-ing.
 */
public enum Activity {
    READ (1),
    WRITE(2),
    EXCEPTION(4);

    private int value;

    Activity(int value) {
        this.value = value;
    }

    /**
     * Returns the integer value of this enumeration literal.
     * return the integer value.
     */
    public int getValue() {
        return value;
    }

    /**
     * Returns the result of or-ing this flag into an integer.
     * param val the value to OR this enumeration value into.
```

```
 * return the result of or-ing this flag into the integer parameter.
 */
public int or(int val) {
    return val | value;
}

/**
 * Returns the result of or-ing this flag into another.
 * param val the value to OR this enumeration value into.
 * return the result of or-ing this flag into the integer parameter.
 */
public int or(Activity val) {
    return val.value | value;
}

/**
 * Tests for the presence of this flag in the specified value.
 * param val the value.
 * return <code>true</code> if this flag is present.
 */
public boolean isSet(int val) {
    if (value == val) {
        // Also tests for 0 (NONE) which is assumed to be set only when
        // no other values are set.
        return true;
    }
    return (val & value) != 0;
}
}
```

4.5.5   StreamState

```
package org.ogf.saga.stream;

/**
 * SAGA Stream states.
 * Newly created streams start in the NEW state. A {link Stream#connect()}
 * call brings it either in state OPEN or ERROR. From the OPEN state, when an
 * error occurs it goes to state ERROR. When the remote party closes the
 * connection it goes into state DROPPED, and after a {link Stream#close()}
 * call it goes into state CLOSED.
 * CLOSED, DROPPED, and ERROR are final states: I/O is no longer possible.
 */
public enum StreamState {
    NEW (1),
    OPEN (2),
    CLOSED (3),
    DROPPED (4),
```

```
    ERROR (5);

    private int value;

    StreamState(int value) {
        this.value = value;
    }

    /**
     * Returns the integer value of this enumeration literal.
     * return the integer value.
     */
    public int getValue() {
        return value;
    }
}
```

4.6 SAGA Remote Procedure Call

GridRPC is one of the few high level APIs that have been specified by the GGF [8]. Semantically, the methods defined in the GridRPC specification, map exactly with the RPC package of the SAGA API as described here. In essence, the GridRPC API has been imported into the SAGA RPC package, and has been equipped with the Look-&-Feel, error conventions, task model, etc. of the SAGA API.

4.6.1 RPC

```
package org.ogf.saga.rpc;

import org.ogf.saga.SagaObject;
import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.DoesNotExist;
import org.ogf.saga.error.IncorrectState;
import org.ogf.saga.error.IncorrectURL;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
import org.ogf.saga.error.Timeout;
import org.ogf.saga.permissions.Permissions;
import org.ogf.saga.task.Async;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;

/**
 * The <code>RPC</code> class represents a remote function handle
 * that can be called repeatedly.
 */
public interface RPC extends SagaObject, Async, Permissions {

    /**
     * Calls the remote procedure.
     * param parameters arguments and results for the call.
     * exception IncorrectURL may be thrown here because the RPC server
     *     that was specified to the factory may not have been contacted
     *     before invoking the call.
     * exception NoSuccess is thrown for arbitrary backend failures, with
     *     a descriptive error message.
     */
    public void call(Parameter... parameters)
        throws NotImplemented, IncorrectURL,
            AuthenticationFailed, AuthorizationFailed,
```

```
        PermissionDenied, BadParameter, IncorrectState, DoesNotExist,
        Timeout, NoSuccess;

/**
 * Non-blocking close of the RPC handle instance.
 * Note for Java implementations: A finalizer could be used in case
 * the application forgets to close.
 */
public void close()
    throws NotImplemented, IncorrectState, NoSuccess;

/**
 * Closes the RPC handle instance.
 * Note for Java implementations: A finalizer could be used in case
 * the application forgets to close.
 * param timeoutInSeconds seconds to wait.
 */
public void close(float timeoutInSeconds)
    throws NotImplemented, IncorrectState, NoSuccess;

//
// Task versions ...
//

/**
 * Creates a task for calling the remote procedure.
 * param mode the task mode.
 * param parameters arguments and results for the call.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task call(TaskMode mode, Parameter... parameters)
    throws NotImplemented;

/**
 * Creates a task for closing the RPC handle instance.
 * param mode the task mode.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public Task close(TaskMode mode)
    throws NotImplemented;

/**
 * Creates a task for closing the RPC handle instance.
 * param mode the task mode.
 * param timeoutInSeconds seconds to wait.
 * return the task.
```

```
    * exception NotImplemented is thrown when the task version of this
    *      method is not implemented.
    */
   public Task close(TaskMode mode, float timeoutInSeconds)
       throws NotImplemented;
}
```

4.6.2   Parameter

```
package org.ogf.saga.rpc;

import org.ogf.saga.buffer.Buffer;

/**
 * Extends the {link Buffer} interface with methods to set/get the modus
 * of RPC parameters.
 */
public interface Parameter extends Buffer {

    /**
     * Sets the io mode.
     * param mode the value for io mode.
     */
    public void setIOMode(IOMode mode);

    /**
     * Retrieves the current value for io mode.
     * return the value of io mode.
     */
    public IOMode getIOMode();
}
```

4.6.3   RPCFactory

```
package org.ogf.saga.rpc;

import org.ogf.saga.URL;
import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.AuthenticationFailed;
import org.ogf.saga.error.AuthorizationFailed;
import org.ogf.saga.error.BadParameter;
import org.ogf.saga.error.DoesNotExist;
import org.ogf.saga.error.IncorrectURL;
import org.ogf.saga.error.NoSuccess;
import org.ogf.saga.error.NotImplemented;
import org.ogf.saga.error.PermissionDenied;
```

```
import org.ogf.saga.error.Timeout;
import org.ogf.saga.session.Session;
import org.ogf.saga.session.SessionFactory;
import org.ogf.saga.task.TaskMode;
import org.ogf.saga.task.Task;

/**
 * Factory for objects from the RPC package.
 * Note: the createParameter methods can also throw NotImplemented, because the
 * Buffer create methods can. Error in the SAGA specifications???
 */
public abstract class RPCFactory {

    private static RPCFactory factory;

    private static synchronized void initializeFactory() {
        if (factory == null) {
            factory = ImplementationBootstrapLoader.createRPCFactory();
        }
    }

    /**
     * Creates a Parameter object. To be provided by the implementation.
     * param data data to be used.
     * param mode IN, OUT, INOUT.
     * return the parameter.
     */
    protected abstract Parameter doCreateParameter(byte[] data, IOMode mode)
        throws BadParameter, NoSuccess, NotImplemented;

    /**
     * Creates a Parameter object. To be provided by the implementation.
     * param mode IN, OUT, INOUT.
     * return the parameter.
     */
    protected abstract Parameter doCreateParameter(IOMode mode)
        throws BadParameter, NoSuccess, NotImplemented;

    /**
     * Creates a Parameter object. To be provided by the implementation.
     * param sz the size of the buffer.
     * param mode IN, OUT, INOUT.
     * return the parameter.
     */
    protected abstract Parameter doCreateParameter(int sz, IOMode mode)
        throws BadParameter, NoSuccess, NotImplemented;

    /**
     * Creates a RPC handle instance. To be provided by the implementation.
     * param session the session handle.
```

```
 * param funcname specification of the remote procedure.
 * return the RPC handle instance.
 */
protected abstract RPC doCreateRPC(Session session, URL funcname)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
        BadParameter, DoesNotExist, Timeout, NoSuccess;


/**
 * Creates a task that creates a RPC handle instance.
 * To be provided by the implementation.
 * param mode the task mode.
 * param session the session handle.
 * param funcname specification of the remote procedure.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
protected abstract Task<RPC> doCreateRPC(TaskMode mode,
        Session session, URL funcname)
    throws NotImplemented;


/**
 * Creates a Parameter object. If the mode indicates an Out parameter,
 * <code>data</code> may be <code>null</code>.
 * param data data to be used.
 * param mode IN, OUT, INOUT.
 * return the parameter.
 * throws NotImplemented
 */
public static Parameter createParameter(byte[] data, IOMode mode)
    throws BadParameter, NoSuccess, NotImplemented {
    initializeFactory();
    return factory.doCreateParameter(data, mode);
}


/**
 * Creates a Parameter object. To be provided by the implementation.
 * param mode IN, OUT, INOUT.
 * return the parameter.
 * throws NotImplemented
 */
public static Parameter createParameter(IOMode mode)
    throws BadParameter, NoSuccess, NotImplemented {
    initializeFactory();
    return factory.doCreateParameter(mode);
}


/**
 * Creates a Parameter object. To be provided by the implementation.
```

```
 * param sz the size of the buffer.
 * param mode IN, OUT, INOUT.
 * return the parameter.
 * throws NotImplemented
 */
public static Parameter createParameter(int sz, IOMode mode)
    throws BadParameter, NoSuccess, NotImplemented {
    initializeFactory();
    return factory.doCreateParameter(sz, mode);
}

/**
 * Creates an IN Parameter object.
 * param data data to be used.
 * return the parameter.
 * throws NotImplemented
 */
public static Parameter createParameter(byte[] data)
    throws BadParameter, NoSuccess, NotImplemented {
    initializeFactory();
    return factory.doCreateParameter(data, IOMode.IN);
}

/**
 * Creates an IN Parameter object.
 * return the parameter.
 * throws NotImplemented
 */
public static Parameter createParameter()
    throws BadParameter, NoSuccess, NotImplemented {
    initializeFactory();
    return factory.doCreateParameter(IOMode.IN);
}

/**
 * Creates an IN Parameter object.
 * param sz the size of the buffer.
 * return the parameter.
 * throws NotImplemented
 */
public static Parameter createParameter(int sz)
    throws BadParameter, NoSuccess, NotImplemented {
    initializeFactory();
    return factory.doCreateParameter(sz, IOMode.IN);
}


/**
 * Creates a RPC handle instance.
 * param session the session handle.
```

```
 * param funcname specification of the remote procedure.
 * return the RPC handle instance.
 */
public static RPC createRPC(Session session, URL funcname)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
            BadParameter, DoesNotExist, Timeout, NoSuccess {
    initializeFactory();
    return factory.doCreateRPC(session, funcname);
}


/**
 * Creates a task that creates a RPC handle instance.
 * param mode the task mode.
 * param session the session handle.
 * param funcname specification of the remote procedure.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 */
public static Task<RPC> createRPC(TaskMode mode, Session session,
        URL funcname) throws NotImplemented {
    initializeFactory();
    return factory.doCreateRPC(mode, session, funcname);
}


/**
 * Creates a RPC handle instance using the default session.
 * param funcname specification of the remote procedure.
 * return the RPC handle instance.
 */
public static RPC createRPC(URL funcname)
    throws NotImplemented, IncorrectURL,
        AuthenticationFailed, AuthorizationFailed, PermissionDenied,
            BadParameter, DoesNotExist, Timeout, NoSuccess {
    Session session = SessionFactory.createSession();
    initializeFactory();
    return factory.doCreateRPC(session, funcname);
}


/**
 * Creates a task that creates a RPC handle instance using the default session.
 * param mode the task mode.
 * param funcname specification of the remote procedure.
 * return the task.
 * exception NotImplemented is thrown when the task version of this
 *     method is not implemented.
 * exception NoSuccess is thrown when the default session could not be
 *     created.
 */
```

```
    public static Task<RPC> createRPC(TaskMode mode,URL funcname)
            throws NotImplemented, NoSuccess {
        Session session = SessionFactory.createSession();
        initializeFactory();
        return factory.doCreateRPC(mode, session, funcname);
    }
}
```

4.6.4   IOMode

```
package org.ogf.saga.rpc;

/**
 * Describes parameter modes.
 */
public enum IOMode {
    IN (1),
    OUT (2),
    INOUT (3);

    private int value;

    IOMode(int value) {
        this.value = value;
    }

    public int getValue() {
        return this.value;
    }
}
```

5 Intellectual Property Issues

5.1 Contributors

This document is the result of the joint efforts of many contributors. The authors listed here and on the title page are those taking responsibility for the content of the document, and all errors. The editors (underlined) are committed to taking permanent stewardship for this document and can be contacted in the future for inquiries.

Ceriel Jacobs	**Thilo Kielmann**
ceriel@cs.vu.nl	kielmann@cs.vu.nl
Vrije Universiteit	Vrije Universiteit
Dept. of Computer Science	Dept. of Computer Science
De Boelelaan 1083	De Boelelaan 1083
1081HV Amsterdam	1081HV Amsterdam
The Netherlands	The Netherlands

The initial version of the presented SAGA Java language binding was drafted by a design team. Members of that design team did not necessarily contribute text to the document, but did certainly contribute to its current state, and very much so. Additional to the authors listed above, the following people were members of the design team, in alphabetical order:

Nields Drost (VU), Jason Maassen (VU), Andre Merzky (LSU), Rob V. van Nieuwpoort (VU), Kees Verstoep (VU).

Further, the authors would like to thank all contributors from OGF's SAGA-RG and SAGA-CORE-WG, and other related groups. We would like to acknowledge, in alphabetical order, the contributions of: Pascal Kleijer (NEC)

We would escpecially like to thank for the financial support without which we would have been in a much inferior position to produce this document. Our thanks go to OMII-UK and to the European Commission via the XtreemOS project.

5.2 Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it

represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

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

5.3   Disclaimer

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

5.4   Full Copyright Notice

Appendix

TODO: list errata w.r.t. GFD.90 that have been incorporated here, already?

References

[1] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Standard), Jan. 2005.

[2] S. Bradner. Key Words for Use in RFCs to Indicate Requirement Levels. RFC 2119, Internet Engineering Task Force (IETF), 1997. `http://www.ietf.org/rfc/rfc2119.txt`.

[3] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, A. Merzky, J. Shalf, and C. Smith. A Simple API for Grid Applications (SAGA). Grid Forum Document GFD.90, 2008. Global Grid Forum.

[4] J. Gosling, B. Joy, G. Steele, and G. Bracha. *The Java(TM) Language Specification, 3rd Edition*. Addison Wesley, 2005. `http://java.sun.com/docs/books/jls/third_edition/html/j3TOC.html`.

[5] JSDL Working Group. Open Grid Forum. `http://forge.ogf.org/sf/projects/jsdl-wg/`.

[6] P. Leach, M. Mealling, and R. Salz. A Universally Unique IDentifier (UUID) URN Namespace. RFC 4122, Internet Engineering Task Force (IETF), 2005. `http://www.ietf.org/rfc/rfc4122.txt` .

[7] A. Merzky and S. Jha. A Requirements Analysis for a Simple API for Grid Applications. Grid Forum Document GFD.71, 2006. Global Grid Forum.

[8] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova. A GridRPC Model and API for End-User Applications. Grid Forum Document GFD.52, 2005. Global Grid Forum.

[9] *Portable Operating System Interface (POSIX) – Part 1: System Application Program Interface (API) [C Language]*. Information technology – Portable Operating System Interface (POSIX). IEEE Computer Society, 345 E. 47th St, New York, NY 10017, USA, 1990.

[10] R. V. van Nieuwpoort, T. Kielmann, and H. E. Bal. User-friendly and reliable grid computing based on imperfect middleware. In *ACM/IEEE Conference on Supercomputing (SC'07)*, 2007.