GWD-R.96
SAGA-CORE-WG

Version: 1.0

Ceriel Jacobs, VU
Thilo Kielmann, VU[1]
other contributors??
June 10, 2011

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

Status of This Document

This document provides information to the grid community, proposing the language binding to the Java language for the Simple API for Grid Applications (SAGA), as specified in GFD.90 [3]. It is supposed to be used by both implementors of this binding and by application programmers. Distribution is unlimited.

Copyright Notice

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 the Saga website at `http://saga.cct.lsu.edu` 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 must 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.NotImplementedException;
import org.ogf.saga.file.FileFactory;
import org.ogf.saga.isn.ISNFactory;
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.sd.SDFactory;
import org.ogf.saga.session.SessionFactory;
import org.ogf.saga.stream.StreamFactory;
import org.ogf.saga.task.TaskFactory;
import org.ogf.saga.url.URLFactory;

/**
 * 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.
     */
    BufferFactory createBufferFactory();

    /**
     * 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 NotImplementedException
     *                is thrown when jobs are not implemented.
     */
    JobFactory createJobFactory() throws NotImplementedException;

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

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

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

    /**
     * Creates a factory for the Saga RPC package.
     *
     * return the RPC factory.
     * exception NotImplementedException
     *                is thrown when RPC is not implemented.
     */
    RPCFactory createRPCFactory() throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when streams are not implemented.
     */
    StreamFactory createStreamFactory() throws NotImplementedException;

    /**
     * Creates a factory for the Saga task package.
     *
     * return the task factory.
     * exception NotImplementedException
     *                is thrown when tasks are not implemented.
     */
    TaskFactory createTaskFactory() throws NotImplementedException;

    /**
     * Creates a factory for the Saga URL package.
     *
     * return the URL factory.
     */
    URLFactory createURLFactory();

}
```

2.6.2 ImplementationBootstrapLoader

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

import java.lang.reflect.InvocationTargetException;
import java.util.*;

import org.ogf.saga.buffer.BufferFactory;
import org.ogf.saga.context.ContextFactory;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.file.FileFactory;
import org.ogf.saga.isn.ISNFactory;
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.sd.SDFactory;
import org.ogf.saga.session.SessionFactory;
import org.ogf.saga.stream.StreamFactory;
import org.ogf.saga.task.TaskFactory;
import org.ogf.saga.url.URLFactory;

/**
 * This class allows the user to have one or more {link SagaFactory SagaFactories},
 * one for each Java Saga implementation that is to be used. A {link SagaFactory}
 * must be used to actually create Saga objects. All factory creation methods have an optional
 * parameter, the classname of the Saga factory. When this parameter is not specified,
 * a default Saga factory is used.
 * <p>
 * The classname of the default Saga factory is to be provided by the
 * the environment variable (Java property) <code>saga.factory</code>.
 * <p>
 * The
 * <code>ImplementationBootstrapLoader</code> instantiates
 * exactly one instance of each Saga factory classname. A Saga factory must have a public parameter-le
 * constructor.
 */
public class ImplementationBootstrapLoader {

    /** One ImplementationBootstrapLoader for each SagaFactory class. */
    private static Map<String, ImplementationBootstrapLoader> s_loaders = new HashMap<String, Implemen

    /** The saga factory instance, specific for a Saga implementation. */
    private SagaFactory sagaFactory;

    private BufferFactory bufferFactory;

    private ContextFactory contextFactory;

    private FileFactory fileFactory;

    private ISNFactory ISNFactory;

    private JobFactory jobFactory;

    private LogicalFileFactory logicalFileFactory;

    private MonitoringFactory monitoringFactory;

    private NSFactory NSFactory;

    private RPCFactory RPCFactory;

    private SDFactory SDFactory;
    private SessionFactory sessionFactory;

    private StreamFactory streamFactory;

    private TaskFactory taskFactory;

    private URLFactory URLFactory;

    private static synchronized ImplementationBootstrapLoader getLoader(String factoryName) throws NoS

// Deterimine SagaFactory classname if not specified.
        if (factoryName == null) {
            Properties sagaProperties = SagaProperties.getDefaultProperties();
            // Obtain the name of the SAGA factory.
            factoryName = sagaProperties.getProperty(SagaProperties.FACTORY);
            if (factoryName == null) {
                throw new NoSuccessException("No SAGA factory name specified");
            }
        }

        // See if this factory is already available.
        ImplementationBootstrapLoader loader = s_loaders.get(factoryName);
        if (loader == null) {
            // Nope. Create it.
            loader = new ImplementationBootstrapLoader(factoryName);

            // Put instance in map
            s_loaders.put(factoryName, loader);
        }

        return loader;
    }

    private ImplementationBootstrapLoader(String factoryName) throws NoSuccessException {

        // 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 NoSuccessException("Could not load class "
                    + factoryName, e);
        }

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

    /**
     * Creates a buffer factory, using the specified SagaFactory.
     *
     * param factoryName classname of the Saga factory to be used, or null.
     * return a buffer factory.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static BufferFactory getBufferFactory(String factoryName)
            throws NoSuccessException {
ImplementationBootstrapLoader l = getLoader(factoryName);
synchronized(l) {
    if (l.bufferFactory == null) {
l.bufferFactory = l.sagaFactory.createBufferFactory();
    }
    return l.bufferFactory;
}
    }

    /**
     * Creates a context factory, using the specified SagaFactory.
     *
     * param factoryName classname of the Saga factory to be used, or null.
     * return a context factory.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static ContextFactory getContextFactory(String factoryName)
            throws NoSuccessException {
ImplementationBootstrapLoader l = getLoader(factoryName);
synchronized(l) {
    if (l.contextFactory == null) {
l.contextFactory = l.sagaFactory.createContextFactory();
    }
    return l.contextFactory;
}
```

```
    }

    /**
     * Creates a session factory, using the specified SagaFactory.
     *
     * param factoryName classname of the Saga factory to be used, or null.
     * return a session factory.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static SessionFactory getSessionFactory(String factoryName)
            throws NoSuccessException {
ImplementationBootstrapLoader l = getLoader(factoryName);
synchronized(l) {
    if (l.sessionFactory == null) {
l.sessionFactory = l.sagaFactory.createSessionFactory();
    }
    return l.sessionFactory;
}
    }

    /**
     * Creates a file factory, using the specified SagaFactory.
     * This method annot throw NotImplemented, because the IOVec
     * constructor cannot (according to the SAGA specs).
     *
     * param factoryName classname of the Saga factory to be used, or null.
     * return a file factory.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static FileFactory getFileFactory(String factoryName)
            throws NoSuccessException {
ImplementationBootstrapLoader l = getLoader(factoryName);
synchronized(l) {
    if (l.fileFactory == null) {
l.fileFactory = l.sagaFactory.createFileFactory();
    }
    return l.fileFactory;
}
    }

    /**
     * Creates a job factory, using the specified SagaFactory.
     *
     * param factoryName classname of the Saga factory to be used, or null.
     * return a job factory.
     * throws NotImplementedException
     *             is thrown when jobs are not implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static JobFactory getJobFactory(String factoryName) throws NotImplementedException,
            NoSuccessException {
ImplementationBootstrapLoader l = getLoader(factoryName);
synchronized(l) {
    if (l.jobFactory == null) {
l.jobFactory = l.sagaFactory.createJobFactory();
    }
    return l.jobFactory;
}
    }

    /**
     * Creates a logical file factory, using the specified SagaFactory.
     *
     * param factoryName classname of the Saga factory to be used, or null.
     * return a logical file factory.
     * throws NotImplementedException
     *             is thrown when logical files are not implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static LogicalFileFactory getLogicalFileFactory(String factoryName)
            throws NotImplementedException, NoSuccessException {
ImplementationBootstrapLoader l = getLoader(factoryName);
synchronized(l) {
    if (l.logicalFileFactory == null) {
l.logicalFileFactory = l.sagaFactory.createLogicalFileFactory();
    }
    return l.logicalFileFactory;
}
    }

    /**
     * Creates a monitoring factory, using the specified SagaFactory.
     *
     * param factoryName classname of the Saga factory to be used, or null.
     * return a monitoring factory.
     * throws NotImplementedException
     *             is thrown when monitoring is not implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static MonitoringFactory getMonitoringFactory(String factoryName)
            throws NotImplementedException, NoSuccessException {
ImplementationBootstrapLoader l = getLoader(factoryName);
synchronized(l) {
    if (l.monitoringFactory == null) {
l.monitoringFactory = l.sagaFactory.createMonitoringFactory();
```

```
    }
    return l.monitoringFactory;
}
    }

    /**
     * Creates a namespace factory, using the specified SagaFactory.
     *
     * param factoryName classname of the Saga factory to be used, or null.
     * return a namespace factory.
     * throws NotImplementedException
     *             is thrown when namespace is not implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static NSFactory getNamespaceFactory(String factoryName)
            throws NotImplementedException, NoSuccessException {
ImplementationBootstrapLoader l = getLoader(factoryName);
synchronized(l) {
    if (l.NSFactory == null) {
l.NSFactory = l.sagaFactory.createNamespaceFactory();
    }
    return l.NSFactory;
}
    }

    /**
     * Creates an RPC factory, using the specified SagaFactory.
     *
     * param factoryName classname of the Saga factory to be used, or null.
     * return an RPC factory.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     * throws NotImplementedException
     * throws NotImplementedException
     *             is thrown when RPC is not implemented.
     */
    public static RPCFactory getRPCFactory(String factoryName) throws NoSuccessException,
            NotImplementedException {
ImplementationBootstrapLoader l = getLoader(factoryName);
synchronized(l) {
    if (l.RPCFactory == null) {
l.RPCFactory = l.sagaFactory.createRPCFactory();
    }
    return l.RPCFactory;
}
    }

    /**
     * Creates a stream factory, using the specified SagaFactory.
     *
     * param factoryName classname of the Saga factory to be used, or null.
     * return a stream factory.
     * throws NotImplementedException
     *             is thrown when streams are not implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static StreamFactory getStreamFactory(String factoryName)
            throws NotImplementedException, NoSuccessException {
ImplementationBootstrapLoader l = getLoader(factoryName);
synchronized(l) {
    if (l.streamFactory == null) {
l.streamFactory = l.sagaFactory.createStreamFactory();
    }
    return l.streamFactory;
}
    }

    /**
     * Creates a task factory, using the specified SagaFactory.
     *
     * param factoryName classname of the Saga factory to be used, or null.
     * return a task factory.
     * throws NotImplementedException
     *             is thrown when tasks are not implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static TaskFactory getTaskFactory(String factoryName)
            throws NotImplementedException, NoSuccessException {
ImplementationBootstrapLoader l = getLoader(factoryName);
synchronized(l) {
    if (l.taskFactory == null) {
l.taskFactory = l.sagaFactory.createTaskFactory();
    }
    return l.taskFactory;
}
    }

    /**
     * Creates an URL factory, using the specified SagaFactory.
     *
     * param factoryName classname of the Saga factory to be used, or null.
     * return an URL factory.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static URLFactory getURLFactory(String factoryName)
            throws NoSuccessException {
```

```
ImplementationBootstrapLoader l = getLoader(factoryName);
synchronized(l) {
    if (l.URLFactory == null) {
l.URLFactory = l.sagaFactory.createURLFactory();
    }
    return l.URLFactory;
}
    }

}
```

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

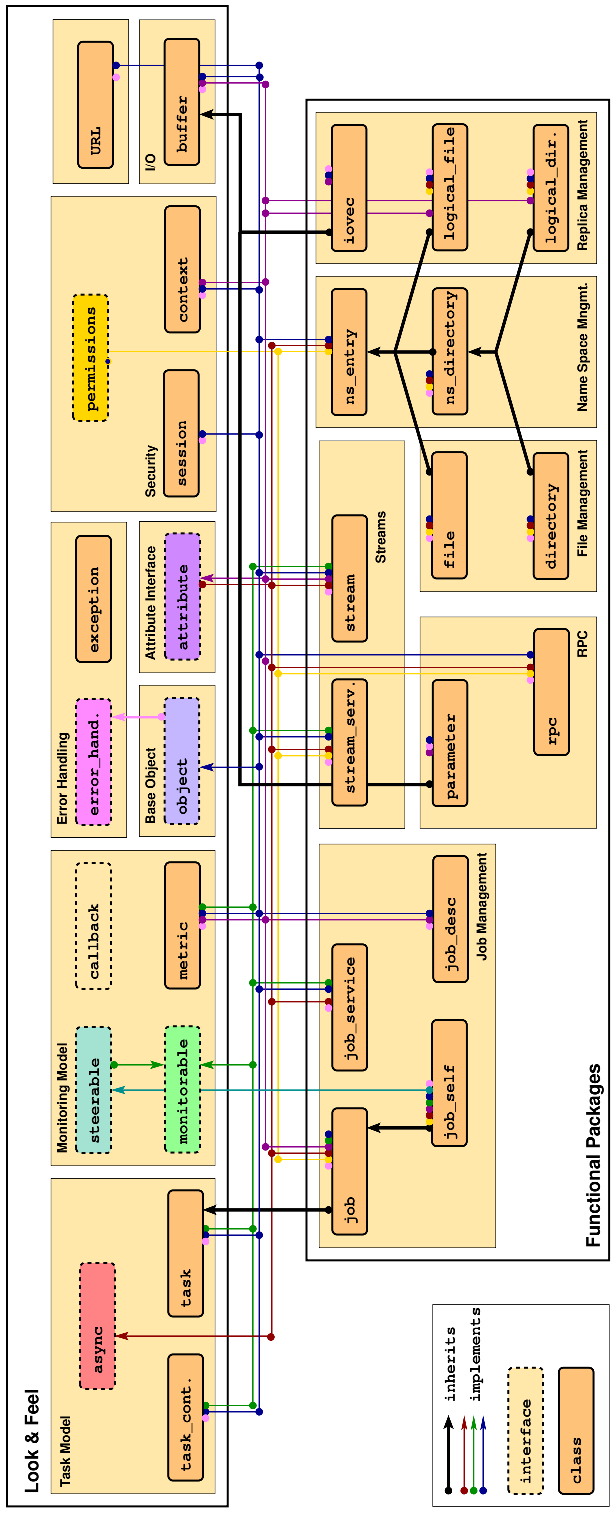

Figure 1: The SAGA class and interface hierarchy, accordign to GFD.90.

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. Whereever the SAGA specifications say that a POSIX `errno` code is to be returned, the Java version throws a `SagaIOException` instead. If a POSIX error code is available, it can be found in the exception object.

A simple mechanism exists for storing and examining exceptions that may be thrown by adaptors in adaptor-based Saga implementations. In such implementations, the top-level exception (the one highest up in the Saga exception hierarchy) is not always the most informative one, and the implementation is not always capable of selecting the most informative exception. In these cases, the implementation may opt to add the individual exceptions as nested exceptions to the exception thrown.

3.1.1 SagaException

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

import java.lang.reflect.Constructor;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.NoSuchElementException;

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.
 * The {link #getMessage} method does not quite behave as specified in the SAGA specs,
 * because Java already has {link #toString} for exactly that behavior.
 * So, {link #getMessage} does what it always does for Java throwables.
 * <p>
 * A simple mechanism exists for storing and examining exceptions that may be thrown
 * by adaptors in adaptor-based Saga implementations. In such implementations, the
 * top-level exception (the one highest up in the Saga exception hierarchy) is not
 * always the most informative one, and the implementation is not always capable
 * of selecting the most informative exception. In these cases, the implementation
 * may opt to add the individual exceptions as nested exceptions to the exception
 * thrown. The nested exceptions can be examined using the {link #getAllExceptions}
 * or {link #getAllMessages} methods.
 * <p>
 * In addition to the {link #getAllExceptions} and {@link #getAllMessages} mechanisms,
 * this implementation also allows the user to iterate over the lower-level exception
 * list.
 */
public abstract class SagaException extends Exception implements
        Comparable<SagaException>, Iterable<SagaException> {

    // Determine the order of the exceptions, most specific first.
    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 INCORRECT_TYPE = 6;

    protected static final int PERMISSION_DENIED = 7;

    protected static final int AUTHORIZATION_FAILED = 8;

    protected static final int AUTHENTICATION_FAILED = 9;

    protected static final int IO_EXCEPTION = 10;

    protected static final int TIMEOUT = 11;

    protected static final int NO_SUCCESS = 12;

    protected static final int NOT_IMPLEMENTED = 13;

    /** 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;

    private final ArrayList<SagaException> nestedExceptions
            = new ArrayList<SagaException>();

    private boolean isCopy = false;

    /**
     * Constructs a new SAGA exception.
     *
     * param order
     *            initializes the exceptionOrder field that determines which
     *            exception is more specific.
     */
    protected SagaException(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.
     */
    protected SagaException(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.
     */
    protected SagaException(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.
     */
    protected SagaException(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.
     */
    protected SagaException(int order, String message, SagaObject object) {
        super(message);
        this.object = object;
        exceptionOrder = order;
    }


    /**
     * Constructs a new SAGA exception with the specified cause and
     * associated SAGA object.
     *
     * param order
     *           initializes the exceptionOrder field that determines which
     *           exception is more specific.
     * param cause
     *           the cause.
     * param object
     *           the SAGA object associated with the exception.
     */
    protected SagaException(int order, Throwable cause, SagaObject object) {
        super(cause);
        this.object = object;
        exceptionOrder = order;
    }

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

    /**
     * Returns the SAGA object associated with this exception.
     * exception DoesNotExistException
     *           is thrown when there is no object associated with this exception.
     * exception NoSuccessException
     *           is thrown when the operation was not successfully performed,
     *           and none of the other exceptions apply.
     * return the associated SAGA object.
     */
    public SagaObject getObject() throws DoesNotExistException,
            NoSuccessException {
        if (object == null) {
            throw new DoesNotExistException(
                    "No object associated with this exception");
        }
        return object;
    }

    /**
     * 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(SagaException o) {
        return exceptionOrder - o.exceptionOrder;
    }

    /**
     * Returns a short description of this <code>SagaException</code>.
     * If this <code>SagaException</code> object was created with a non-null detail message string,
     * then the result is the concatenation of three strings:
     * <ul>
     * <li>The simple (unqualified) name of the actual class of this object
     * <li>": " (a colon and a space)
     * <li>The result of the {link #getMessage} method for this object
     * </ul>
     * If this <code>SagaException</code> object was created with a null detail message string,
     * then only the simple (unqualified) name of the actual class of this object is returned.
     *
     * return a string representation of this <code>SagaException</code>.
     */
    public String toString() {
        String message = getMessage();
        String result = getClass().getSimpleName();
        if (result.endsWith("Exception")) {
            result = result.replace("Exception", "");
        }

        if (message != null) {
            result = result + ": " + message;
        }
        return result;
    }

    /**
     * Adds an exception to the list of nested exceptions. This method should
     * only be used by SAGA implementations.
     * param e the exception to be added to the list.
     */
    public void addNestedException(SagaException e) {
        nestedExceptions.add(e);
    }

    /**
     * Returns an iterator that iterates over the nested exceptions.
     * return the iterator.
     */
    public Iterator<SagaException> iterator() {
        return new ExceptionIterator(this);
    }

    /**
     * Strips nested exceptions from a Saga exception. This is needed for adaptors that
     * call Saga factory methods from within a method, for instance <code>openDir</code>.
     *
     * return the stripped exception.
     */
    private SagaException stripNestedExceptions() {
        try {
            SagaException ex;
            if (this instanceof SagaIOException) {
                Constructor<SagaIOException> c = SagaIOException.class.getConstructor(
                        String.class, Throwable.class, Integer.TYPE,
                        SagaObject.class);

                ex = c.newInstance(getMessage(),
                        getCause(),
                        ((SagaIOException) this).getPosixErrorCode(),
                        object);
            } else {
                Constructor<? extends SagaException> c = getClass().getConstructor(
                        String.class, Throwable.class, SagaObject.class);
                ex = c.newInstance(getMessage(),
                        getCause(), object);
            }
            ex.setStackTrace(getStackTrace());
            ex.isCopy = true;
            return ex;
        } catch (Throwable e) {
            throw new Error("Could not create copy of exception", e);
        }
    }

    /**
     * Gets the list of lower-level exceptions. The first exception in the list
     * is the one on which this method is invoked, but without lower-level exceptions.
     *
     * return the list of exceptions.
     */
    public List<SagaException> getAllExceptions() {
        ArrayList<SagaException> list = new ArrayList<SagaException>();
        if (isCopy) {
            // Method is invoked nested, on a copy.
            // Should return empty list.
            return list;
        }
        list.add(stripNestedExceptions());
        list.addAll(nestedExceptions);
        return list;
    }

    /**
     * Gets the list of lower-level exception messages. The first message in the list
     * is from the one on which this method is invoked.
     *
     * return the list of exception messages.
     */
    public List<String> getAllMessages() {
        ArrayList<String> list = new ArrayList<String>();
        list.add(toString());
        for (SagaException e : nestedExceptions) {
            list.add(e.toString());
        }
        return list;
    }

    private static class ExceptionIterator implements Iterator<SagaException> {
        private SagaException[] exceptions;
        private int index = 0;

        ExceptionIterator(SagaException ex) {
            List<SagaException> list = ex.getAllExceptions();
```

```
            exceptions = list.toArray(new SagaException[list.size()]);
        }

        public boolean hasNext() {
            return index < exceptions.length;
        }

        public SagaException next() {
            if (index < exceptions.length) {
                return exceptions[index++];
            }
            throw new NoSuchElementException("Iterator exhausted");
        }

        public void remove() {
            throw new UnsupportedOperationException("remove() not supported");
        }
    }
}
```

3.1.2 SagaIOException

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

import org.ogf.saga.SagaObject;

/**
 * This exception is specific to the Java Language Bindings for SAGA, which uses
 * this exception for some methods, instead of returning POSIX error code.
 */
public class SagaIOException extends SagaException {

    private static final long serialVersionUID = 1L;

    private final int posixErrorCode;

    /**
     * Constructs an SagaIO exception.
     */
    public SagaIOException() {
        super(IO_EXCEPTION);
        posixErrorCode = 0;
    }

    /**
     * Constructs an SagaIO exception with the specified detail message.
     *
     * param message
     *           the detail message.
     */
    public SagaIOException(String message) {
        super(IO_EXCEPTION, message);
        posixErrorCode = 0;
    }

    /**
     * Constructs an SagaIO exception with the specified detail message and
     * error code,
     *
     * param message
     *           the detail message.
     * param code
     *           the error code.
     */
    public SagaIOException(String message, int code) {
        super(IO_EXCEPTION, message);
        posixErrorCode = code;
    }

    /**
     * Constructs an SagaIO exception with the specified cause.
     *
     * param cause
     *           the cause.
     */
    public SagaIOException(Throwable cause) {
        super(IO_EXCEPTION, cause);
        posixErrorCode = 0;
    }

    /**
     * Constructs an SagaIO exception with the specified detail message and
     * cause.
     *
     * param message
     *           the detail message.
     * param cause
     *           the cause.
     */
    public SagaIOException(String message, Throwable cause) {
        super(IO_EXCEPTION, message, cause);
        posixErrorCode = 0;
    }

    /**
     * Constructs an SagaIO exception with the specified detail message and
     * associated SAGA object.
     *
     * param message
     *           the detail message.
     * param object
     *           the associated SAGA object.
     */
    public SagaIOException(String message, SagaObject object) {
        super(IO_EXCEPTION, message, object);
        posixErrorCode = 0;
    }


    /**
     * Constructs a new SagaIOException with the specified cause and
     * associated SAGA object.
     *
     * param cause
     *           the cause.
     * param object
     *           the SAGA object associated with the exception.
     */
    public SagaIOException(Throwable cause, SagaObject object) {
        super(IO_EXCEPTION, cause, object);
        posixErrorCode = 0;
    }

    /**
     * Constructs a new SagaIOException with the specified detail message,
     * specified cause and associated SAGA object.
     *
     * param detail
     *           the detail message.
     * param cause
     *           the cause.
     * param object
     *           the SAGA object associated with the exception.
     */
    public SagaIOException(String detail, Throwable cause, SagaObject object) {
        super(IO_EXCEPTION, detail, cause, object);
        posixErrorCode = 0;
    }

    /**
     * Constructs an SagaIO exception with the specified detail message. POSIX
     * error code, and associated SAGA object.
     *
     * param message
     *           the detail message.
     * param code
     *           the error code.
     * param cause
     *            the cause.
     * param object
     *            the associated SAGA object.
     */
    public SagaIOException(String message, Throwable cause, int code, SagaObject object) {
        super(IO_EXCEPTION, message, cause, object);
        posixErrorCode = code;
    }

    /**
     * Returns the POSIX error code associated with this exception, in case it
     * is available. If not, this method returns 0.
     *
     * return the error code.
     */
    public int getPosixErrorCode() {
        return posixErrorCode;
    }
}
```

3.1.3 NotImplementedException

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

import org.ogf.saga.SagaObject;

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

    private static final long serialVersionUID = 1L;

    /**
     * Constructs a NotImplemented exception.
     */
    public NotImplementedException() {
        super(NOT_IMPLEMENTED);
    }

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

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

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

}
```

3.1.4 IncorrectURLException

```
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 IncorrectURLException extends SagaException {

    private static final long serialVersionUID = 1L;

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

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

    /**
     * Constructs a new IncorrectURLException with the specified cause and
     * associated SAGA object.
     *
     * param cause
     *           the cause.
     * param object
     *           the SAGA object associated with the exception.
     */
    public IncorrectURLException(Throwable cause, SagaObject object) {
        super(INCORRECT_URL, cause, object);
    }
    /**
     * Constructs a new IncorrectURLException with the specified detail message,
     * specified cause and associated SAGA object.
     *
     * param detail
     *            the detail message.
     * param cause
     *            the cause.
     * param object
     *            the SAGA object associated with the exception.
     */
    public IncorrectURLException(String detail, Throwable cause, SagaObject object) {
        super(INCORRECT_URL, detail, cause, object);
    }

}
```

3.1.5 BadParameterException

```
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 BadParameterException extends SagaException {

    private static final long serialVersionUID = 1L;

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

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

    /**
     * Constructs a new BadParameterException with the specified cause and
     * associated SAGA object.
     *
     * param cause
     *            the cause.
     * param object
     *            the SAGA object associated with the exception.
     */
    public BadParameterException(Throwable cause, SagaObject object) {
        super(BAD_PARAMETER, cause, object);
    }
    /**
     * Constructs a new BadParameterExceptio with the specified detail message,
     * specified cause and associated SAGA object.
     *
     * param detail
     *            the detail message.
     * param cause
     *            the cause.
     * param object
     *            the SAGA object associated with the exception.
     */
    public BadParameterException(String detail, Throwable cause, SagaObject object) {
        super(BAD_PARAMETER, detail, cause, object);
    }

}
```

3.1.6 AlreadyExistsException

```
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 AlreadyExistsException extends SagaException {

    private static final long serialVersionUID = 1L;

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

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

    /**
     * Constructs a new AlreadyExistsException with the specified cause and
     * associated SAGA object.
     *
     * param cause
     *            the cause.
     * param object
     *            the SAGA object associated with the exception.
     */
    public AlreadyExistsException(Throwable cause, SagaObject object) {
        super(ALREADY_EXISTS, cause, object);
    }

    /**
```

```
     * Constructs a new AlreadyExistsException with the specified detail message,
     * specified cause and associated SAGA object.
     *
     * param detail
     *            the detail message.
     * param cause
     *            the cause.
     * param object
     *            the SAGA object associated with the exception.
     */
    public AlreadyExistsException(String detail, Throwable cause, SagaObject object) {
        super(ALREADY_EXISTS, detail, cause, object);
    }

}
```

3.1.7 DoesNotExistException

```
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 DoesNotExistException extends SagaException {

    private static final long serialVersionUID = 1L;

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

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

    /**
```

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


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

}
```

3.1.8 IncorrectStateException

```
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 IncorrectStateException extends SagaException {

    private static final long serialVersionUID = 1L;

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

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

    /**
     * Constructs a new IncorrectStateException with the specified cause and
     * associated SAGA object.
     *
     * param cause
     *            the cause.
     * param object
     *            the SAGA object associated with the exception.
     */
    public IncorrectStateException(Throwable cause, SagaObject object) {
        super(INCORRECT_STATE, cause, object);
    }

    /**
```

```
     * Constructs a new IncorrectStateException with the specified detail message,
     * specified cause and associated SAGA object.
     *
     * param detail
     *           the detail message.
     * param cause
     *           the cause.
     * param object
     *           the SAGA object associated with the exception.
     */
    public IncorrectStateException(String detail, Throwable cause, SagaObject object) {
        super(INCORRECT_STATE, detail, cause, object);
    }

}
```

3.1.9 PermissionDeniedException

```
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 PermissionDeniedException extends SagaException {

    private static final long serialVersionUID = 1L;

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

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

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

}
```

3.1.10 AuthorizationFailedException

```
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 AuthorizationFailedException extends SagaException {

    private static final long serialVersionUID = 1L;

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

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

    /**
     * Constructs a new AuthorizationFailedException with the specified cause and
     * associated SAGA object.
     *
     * param cause
     *            the cause.
     * param object
     *            the SAGA object associated with the exception.
     */
    public AuthorizationFailedException(Throwable cause, SagaObject object) {
        super(AUTHORIZATION_FAILED, cause, object);
    }
    /**
     * Constructs a new AuthorizationFailedException with the specified detail message,
     * specified cause and associated SAGA object.
     *
     * param detail
     *            the detail message.
     * param cause
     *            the cause.
     * param object
     *            the SAGA object associated with the exception.
     */
    public AuthorizationFailedException(String detail, Throwable cause, SagaObject object) {
        super(AUTHORIZATION_FAILED, detail, cause, object);
    }

}
```

3.1.11 AuthenticationFailedException

```
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 AuthenticationFailedException extends SagaException {

    private static final long serialVersionUID = 1L;

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

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

    /**
     * Constructs a new AuthenticationFailedException with the specified cause and
     * associated SAGA object.
     *
     * param cause
     *            the cause.
     * param object
     *            the SAGA object associated with the exception.
     */
    public AuthenticationFailedException(Throwable cause, SagaObject object) {
        super(AUTHENTICATION_FAILED, cause, object);
    }
    /**
     * Constructs a new AuthenticationFailedException with the specified detail message,
     * specified cause and associated SAGA object.
     *
     * param detail
     *            the detail message.
     * param cause
     *            the cause.
     * param object
     *            the SAGA object associated with the exception.
     */
    public AuthenticationFailedException(String detail, Throwable cause, SagaObject object) {
        super(AUTHENTICATION_FAILED, detail, cause, object);
    }

}
```

3.1.12 TimeoutException

```
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 TimeoutException extends SagaException {

    private static final long serialVersionUID = 1L;

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

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

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

}
```

3.1.13 NoSuccessException

```
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 NoSuccessException extends SagaException {

    private static final long serialVersionUID = 1L;

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

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

The language-independent SAGA specification names this object "object", which according to the Java conventions would become "Object". To avoid confusion with `java.lang.Object`, we opted to name this base object "SagaObject".

The `ObjectType` as specified in the language-independent SAGA specification is not included in the Java language bindings. Instead, the Java operator `instanceof` is to be used.

3.2.1 SagaObject

```
package org.ogf.saga;

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

/**
 * 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 DoesNotExistException
     *                is thrown when this method is called on objects that do
     *                not have a session attached.
     */
    public Session getSession() throws DoesNotExistException;

    /**
     * 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;

}
```

3.3 SAGA URLs

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

In respect to the URL problem (stated in GFD.90, Section 2.11), the interface provides 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 URLFactory

```
package org.ogf.saga.url;

import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.NoSuccessException;

/**
 * Factory for URLs.
 */
public abstract class URLFactory {

    private static URLFactory getFactory(String sagaFactoryName)
     throws NoSuccessException {
return ImplementationBootstrapLoader.getURLFactory(sagaFactoryName);
    }

    protected abstract URL doCreateURL(String url)
            throws BadParameterException, NoSuccessException;

    /**
     * Creates an URL object from the specified string.
     * param
     *      url the URL as a string.
     * exception BadParameterException
     *      is thrown when there is a syntax error in the parameter.
     * exception NoSuccessException
     *      is thrown if the implementation cannot create valid
     *      default values based on the available information.
     */
    public static URL createURL(String url)
            throws BadParameterException, NoSuccessException {
        return createURL(null, url);
    }

    /**
     * Creates an URL object from the specified string.
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param
     *      url the URL as a string.
     * exception BadParameterException
     *      is thrown when there is a syntax error in the parameter.
     * exception NoSuccessException
     *      is thrown if the implementation cannot create valid
     *      default values based on the available information.
     */
    public static URL createURL(String sagaFactoryClassname, String url)
            throws BadParameterException, NoSuccessException {
        return getFactory(sagaFactoryClassname).doCreateURL(url);
    }

}
```

3.3.2 URL

```
package org.ogf.saga.url;

import org.ogf.saga.SagaObject;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.session.Session;

/**
 * The <code>URL</code> interface provides methods to access or set
 * the individual parts of an URL, and to convert strings to URLs and
 * vice versa.
 */
public interface URL extends SagaObject {

    /**
     * Replaces the current value of the URL with the specified value.
     *
     * param url
     *           the string.
     * exception BadParameterException
     *           is thrown when there is a syntax error in the parameter.
     */
    public void setString(String url) throws BadParameterException;

    /**
     * Replaces the current value of the URL with "".
     *
     * exception BadParameterException
     *           is thrown by {link #setString(String)}.
     */
    public void setString() throws BadParameterException;

    /**
     * Returns this URL as a string. The result may contain non-escaped characters,
     * so may not be suitable for creating a new URL object. For that, you have to
     * use {link #getEscaped()}.
     *
     * return the string.
     */
    public String getString();


    /**
     * Returns this URL as a string, with escapes added where needed to parse the
     * result as an URL.
     *
     * return the string.
     */
    public String getEscaped();

    /**
     * Returns the fragment part of this URL.
     *
     * return the fragment.
     */
    public String getFragment();

    /**
     * Sets the fragment part of this URL to the specified parameter.
     *
     * param fragment
     *           the fragment.
     * exception BadParameterException
     *           is thrown when there is a syntax error in the parameter.
     */
    public void setFragment(String fragment) throws BadParameterException;
    /**
     * Sets the fragment part of this URL to "".
     * exception BadParameterException
     *                is thrown by {link #setFragment(String)}.
     */
    public void setFragment() throws BadParameterException;

    /**
     * Returns the host part of this URL.
     *
     * return the host.
     */
    public String getHost();

    /**
     * Sets the host part of this URL to the specified parameter.
     *
     * param host
     *            the host.
     * exception BadParameterException
     *            is thrown when there is a syntax error in the parameter.
     */
    public void setHost(String host) throws BadParameterException;

    /**
     * Sets the host part of this URL to "".
     * exception BadParameterException
     *            is thrown by {link #setHost(String)}.
     */
    public void setHost() throws BadParameterException;

    /**
     * Returns the path part of this URL.
     *
     * return the path.
     */
    public String getPath();

    /**
     * Sets the path part of this URL to the specified parameter.
     *
     * param path
     *            the path.
     * exception BadParameterException
     *            is thrown when there is a syntax error in the path.
     */
    public void setPath(String path) throws BadParameterException;


    /**
```

```
     * Sets the path part of this URL to "".
     * exception BadParameterException
     *            is thrown by {link #setPath(String)}.
     */
    public void setPath() throws BadParameterException;

    /**
     * Returns the port number of this URL.
     *
     * return the port number.
     */
    public int getPort();

    /**
     * Sets the port number of this URL to the specified parameter.
     *
     * param port
     *            the port number.
     * exception BadParameterException
     *            is thrown when there is an error in the parameter.
     */
    public void setPort(int port) throws BadParameterException;


    /**
     * Sets the port number of this URL to -1.
     * exception BadParameterException
     *            is thrown by {link #setPort()}.
     */
    public void setPort() throws BadParameterException;

    /**
     * Returns the query part from this URL.
     *
     * return the query.
     */
    public String getQuery();

    /**
     * Sets the query part of this URL to the specified parameter.
     *
     * param query
     *            the query.
     * exception BadParameterException
     *            is thrown when there is a syntax error in the parameter.
     */
    public void setQuery(String query) throws BadParameterException;

    /**
     * Sets the query part of this URL to "".
     * exception BadParameterException
     *            is thrown by {link #setQuery(String)}.
     */
    public void setQuery() throws BadParameterException;

    /**
     * Returns the scheme part from this URL.
     *
     * return the scheme.
     */
    public String getScheme();

    /**
     * Sets the scheme part of this URL to the specified parameter.
     *
     * param scheme
     *            the scheme.
     * exception BadParameterException
     *            is thrown when there is a syntax error in the parameter.
     */
    public void setScheme(String scheme) throws BadParameterException;


    /**
     * Sets the scheme part of this URL to the specified parameter.
     * exception BadParameterException
     *            is thrown by {link #setScheme(String)}.
     */
    public void setScheme() throws BadParameterException;

    /**
     * Returns the userinfo part from this URL.
     *
     * return the userinfo.
     */
    public String getUserInfo();

    /**
     * Sets the user info part of this URL to the specified parameter.
     *
     * param userInfo
     *            the userinfo.
     * exception BadParameterException
     *            is thrown when there is a syntax error in the parameter.
     */
    public void setUserInfo(String userInfo) throws BadParameterException;


    /**
     * Sets the user info part of this URL to "".
     * exception BadParameterException
     *            is thrown by {link #setUserInfo(String)}.
     */
    public void setUserInfo() throws BadParameterException;

    /**
     * Returns a new URL with the scheme part replaced.
     *
     * param scheme
     *            the new scheme.
     * return the new URL.
     * exception BadParameterException
     *            is thrown when there is a syntax error in the new URL.
     * exception NoSuccessException
     *            is thrown when the scheme is supported, but the URL
     *            cannot be translated to the scheme.
     */
    public URL translate(String scheme) throws BadParameterException, NoSuccessException;


    /**
     * Returns a new URL with the scheme part replaced.
     *
     * param session
     *            session to be used for possibly-needed back-end communication.
     * param scheme
     *            the new scheme.
     * return the new URL.
     * exception BadParameterException
     *            is thrown when there is a syntax error in the new URL.
     * exception NoSuccessException
     *            is thrown when the scheme is supported, but the URL
     *            cannot be translated to the scheme.
     */
    public URL translate(Session session, String scheme) throws BadParameterException,
            NoSuccessException;

    /**
     * See {link java.net.URI#resolve(java.net.URI)}.
     *
     * param url
     *            the url to resolve with respect to this one.
     * return the resolved url.
     * exception NoSuccessException
     *            is thrown when resolving fails for some reason.
     */
    public URL resolve(URL url) throws NoSuccessException;

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

    /**
     * See {link java.net.URI#normalize()}.
     *
     * return a normalized URL.
     */
    public URL normalize();

}
```

3.4 SAGA I/O Buffer

The SAGA API includes a number of calls which perform byte-level I/O operations, 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 methods. 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.

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, or nothing, in which case the buffer is implementation-managed, and its size determined by the operation at hand.

3.4.1 Buffer

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

import org.ogf.saga.SagaObject;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.DoesNotExistException;
import org.ogf.saga.error.IncorrectStateException;
import org.ogf.saga.error.NoSuccessException;

/**
 * Encapsulates a sequence of bytes to be used for I/O operations.
 */
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.
     * exception IncorrectStateException
     *      is thrown when the buffer is closed.
     * exception BadParameterException
     *      is thrown when the implementation cannot handle the specified size.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void setSize(int size) throws BadParameterException,
            IncorrectStateException, NoSuccessException;

    /**
     * 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.
     * exception IncorrectStateException
     *      is thrown when the buffer is closed.
     * exception BadParameterException
     *      is thrown when the implementation cannot handle the default size.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void setSize() throws BadParameterException,
            IncorrectStateException, NoSuccessException;

    /**
     * Retrieves the current value of the buffer size.
     *
     * return
     *      the size.
     * exception IncorrectStateException
     *      is thrown when the buffer is closed.
     */
    public int getSize() throws IncorrectStateException;

    /**
     * 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.
     * exception IncorrectStateException
     *      is thrown when the buffer is closed.
     * exception BadParameterException
     *      is thrown when the implementation cannot handle the specified data buffer.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void setData(byte[] data) throws BadParameterException,
            IncorrectStateException, NoSuccessException;

    /**
     * Retrieves the buffer data.
     *
     * return
     *      the data.
     * exception IncorrectStateException
     *      is thrown when the buffer is closed.
     * exception DoesNotExistException
     *      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 DoesNotExistException, IncorrectStateException;

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

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

3.4.2 BufferFactory

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

import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.NoSuccessException;

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

    private static BufferFactory getFactory(String sagaFactoryName)
            throws NoSuccessException {
        return ImplementationBootstrapLoader.getBufferFactory(sagaFactoryName);
    }

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


    /**
     * 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 BadParameterException, NoSuccessException;

    /**
     * Creates a (application-allocated) buffer. The size is implicit in the
     * size of the specified array.
     *
     * param data
     *      the storage.
     * return
     *      the buffer.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception BadParameterException
     *      is thrown when the implementation cannot handle the specified data buffer.
     */
    public static Buffer createBuffer(byte[] data)
            throws BadParameterException, NoSuccessException {
return createBuffer(null, data);
    }

    /**
     * Creates a (application-allocated) buffer. The size is implicit in the
     * size of the specified array.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param data
     *      the storage.
     * return
     *      the buffer.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception BadParameterException
     *      is thrown when the implementation cannot handle the specified data buffer.
     */
    public static Buffer createBuffer(String sagaFactoryClassname, byte[] data)
            throws BadParameterException, NoSuccessException {
BufferFactory factory = getFactory(sagaFactoryClassname);
        return factory.doCreateBuffer(data);
    }

    /**
     * Creates a (implementation-managed) buffer of the specified size.
     *
     * param size
     *      the size.
     * return
     *      the buffer.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception BadParameterException
     *      is thrown when the implementation cannot handle the specified size.
     */
    public static Buffer createBuffer(int size) throws BadParameterException,
            NoSuccessException {
return createBuffer(null, size);
    }


    /**
     * Creates a (implementation-managed) buffer of the specified size.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param size
     *      the size.
     * return
     *      the buffer.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception BadParameterException
     *      is thrown when the implementation cannot handle the specified size.
     */
    public static Buffer createBuffer(String sagaFactoryClassname, int size) throws BadParameterExcept
            NoSuccessException {
BufferFactory factory = getFactory(sagaFactoryClassname);
        return factory.doCreateBuffer(size);
    }

    /**
     * Creates a (implementation-managed) buffer.
     *
     * return
     *      the buffer.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception BadParameterException
     *      is thrown when the defaults are not suitable.
     */
    public static Buffer createBuffer() throws BadParameterException,
            NoSuccessException {
return createBuffer((String)null);
    }

    /**
```

```
     * Creates a (implementation-managed) buffer.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * return
     *      the buffer.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception BadParameterException
     *      is thrown when the defaults are not suitable.
     */
    public static Buffer createBuffer(String sagaFactoryClassname) throws BadParameterException,
            NoSuccessException {
        BufferFactory factory = getFactory(sagaFactoryClassname);
        return factory.doCreateBuffer(-1);
    }
}
```

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.DoesNotExistException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.TimeoutException;

/**
 * A session isolates independent sets of SAGA objects from each other, and
 * supports 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.
     * exception NoSuccessException
     *      is thrown when the implementation is not able to initialize
     *      the context, and cannot use the context as-is.
     * exception TimeoutException
     *      is thrown if the context initialization implies a remote operation,
     *      and that operation times out.
     */
    public void addContext(Context context) throws NoSuccessException, TimeoutException;

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

    /**
```

```
     * Retrieves all contexts attached to the session. An empty array is
     * returned if no context is attached. The contexts in the returned list
     * are deep copies of the session's context.
     *
     * return
     *      a list of contexts.
     */
    public Context[] listContexts();

    /**
     * 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();

    /**
     * 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);
}
```

3.5.2 SessionFactory

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

import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.NoSuccessException;

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

    private static SessionFactory getFactory(String sagaFactoryName)
     throws NoSuccessException {
return ImplementationBootstrapLoader.getSessionFactory(sagaFactoryName);
    }

    /**
     * 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 NoSuccessException;

    /**
     * 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.
     * exception NoSuccessException
     *      is thrown if the implementation cannot create valid
     *      default values based on the available information.
     */
    public static Session createSession(boolean defaults)
            throws NoSuccessException {
return createSession(null, defaults);
    }

    /**
     * Creates a session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * 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.
     * exception NoSuccessException
     *      is thrown if the implementation cannot create valid
     *      default values based on the available information.
     */
    public static Session createSession(String sagaFactoryClassname, boolean defaults)
            throws NoSuccessException {
return getFactory(sagaFactoryClassname).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.
     * exception NoSuccessException
     *      is thrown if the implementation cannot create valid
     *      default values based on the available information.
     */
    public static Session createSession() throws NoSuccessException {
        return createSession(true);
    }

    /**
     * Returns the default session, with all the default contexts. Later
     * modifications to this session are reflected in the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * return
     *      the session.
     * exception NoSuccessException
     *      is thrown if the implementation cannot create valid
     *      default values based on the available information.
     */
    public static Session createSession(String sagaFactoryClassname) throws NoSuccessException {
        return createSession(sagaFactoryClassname, 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.

3.6.1 Context

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

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

/**
 * 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";

}
```

3.6.2 ContextFactory

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

import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.IncorrectStateException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.TimeoutException;

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

    /**
     * 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 IncorrectStateException, TimeoutException, NoSuccessException;

    private static ContextFactory getFactory(String sagaFactoryName) throws NoSuccessException {
        return ImplementationBootstrapLoader.getContextFactory(sagaFactoryName);
    }

    /**
     * Constructs a security context.
     *
     * param type
     *      type of the context.
     * return
     *      the security context.
     * exception IncorrectStateException
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown if the implementation cannot create valid
     *      default values based on the available information.
     */
    public static Context createContext(String type)
            throws IncorrectStateException, TimeoutException, NoSuccessException {
return createContext(null, type);
    }

    /**
     * Constructs a security context.
     *
     * return the security context.
     * exception IncorrectStateException
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown if the implementation cannot create valid
     *      default values based on the available information.
     */
    public static Context createContext() throws IncorrectStateException,
            TimeoutException, NoSuccessException {
        return createContext("");
    }

    /**
```

```
     * Constructs a security context.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param type
     *      type of the context.
     * return
     *      the security context.
     * exception IncorrectStateException
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown if the implementation cannot create valid
     *      default values based on the available information.
     */
    public static Context createContext(String sagaFactoryClassname, String type)
            throws IncorrectStateException, TimeoutException, NoSuccessException {
        return getFactory(sagaFactoryClassname).doCreateContext(type);
    }
}
```

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.

The possible permission flags are defined in the `Permission` enumeration class, which includes methods to combine them into integers, and to determine if they are set in an integer. These methods are not described in the language-independent SAGA specification, but are added in the Java language bindings because in Java, enumerations cannot be treated as integers.

3.7.1 Permissions

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

import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;
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. The generic
 * type <code>T</code> specifies the object type implementing the permissions.
 * The permissions are described in the {link Permission} enumeration
 * class.
 */
public interface Permissions<T> 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the given id is unknown or not supported.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void permissionsAllow(String id, int permissions)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the given id is unknown or not supported.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void permissionsDeny(String id, int permissions)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the given id is unknown or not supported.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public boolean permissionsCheck(String id, int permissions)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, TimeoutException, NoSuccessException;

    /**
     * Gets the owner id of the entity.
     *
     * return
     *      the id of the owner.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public String getOwner() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, TimeoutException, NoSuccessException;

    /**
```

```
     * Gets the group id of the entity.
     *
     * return
     *     the id of the group.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public String getGroup() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, TimeoutException, NoSuccessException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<T, Void> permissionsAllow(TaskMode mode, String id,
            int permissions) throws NotImplementedException;
```

```
/**
 * 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 NotImplementedException
 *               is thrown when the task version of this method is not
 *               implemented.
 */
public Task<T, Void> permissionsDeny(TaskMode mode, String id,
        int permissions) throws NotImplementedException;

/**
 * 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 NotImplementedException
 *               is thrown when the task version of this method is not
 *               implemented.
 */
public Task<T, Boolean> permissionsCheck(TaskMode mode, String id,
        int permissions) throws NotImplementedException;

/**
 * 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 NotImplementedException
 *               is thrown when the task version of this method is not
 *               implemented.
 */
public Task<T, String> getOwner(TaskMode mode)
        throws NotImplementedException;

/**
```

```
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<T, String> getGroup(TaskMode mode)
            throws NotImplementedException;
}
```

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 {
    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.

The Java language bindings provide a separate interface `AsyncAttributes` for objects that should implement both `Attributes` and the `Async` interface as specified in the language-independent SAGA specifications. These objects must also provide async versions of the methods in `Attributes`. These async versions are specified in this separate interface.

3.8.1 Attributes

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

import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.DoesNotExistException;
import org.ogf.saga.error.IncorrectStateException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;

/**
 * Provides a uniform paradigm to set and query parameters and properties of
 * SAGA objects. Attributes map a key to a value, and have a type.
 * All keys are represented as a String, values are either represented as
 * a String or a list of Strings, depending on whether the attribute is scalar
 * or a vector. Also, attribute values have a type, which determines how the
 * values are interpreted.
 */
public interface Attributes {

    /** Values of <code>String</code> type attributes are expressed as-is. */
    public static final String STRING = "String";

    /**
     * Values of <code>Int</code> (i.e. Integer) type attributes are expressed
     * as they would in result of a printf of the format %lld as defined by
     * POSIX.
     */
    public static final String INT = "Int";

    /**
     * Values of <code>Enum</code> type attributes are expressed as strings, and
     * have the literal value of the respective enums as defined in the
     * attribute type at hand.
     */
    public static final String ENUM = "Enum";

    /**
     * Values of <code>Float</code> (i.e. floating point) type attributes
     * are expressed as they would in result of a printf of the format
     * %Lf as defined by POSIX.
     */
    public static final String FLOAT = "Float";

    /**
     * Values of <code>Bool</code> type attributes are expressed as
     * 'True' or 'False', as defined below.
     */
    public static final String BOOL = "Bool";

    /**
     * Values of <code>Time</code> type attributes are expressed as they
     * would in result of a call to ctime(), as defined by POSIX, which is almost
     * the same as the result {link java.text.SimpleDateFormat#format(java.util.Date)},
     * when the <code>SimpleDateFormat</code> object is instantiated with
     * "EEE MMM dd HH:mm:ss yyyy". The only difference is that the date field should
     * be space-padded, not zero-padded.
     * Applications can also specify these attribute values as seconds since epoch
     * (this formats the string as an <code>Int</code> type), but all time attributes
     * set by implementations must be in the format described above.
     */
    public static final String TIME = "Time";

    /**
     * <code>Trigger</code> type attributes don't have a value at all.
     * They are used in monitoring.
     */
    public static final String TRIGGER = "Trigger";

    /** 'True' value of a 'Bool' attribute. */
    public static final String TRUE = "True";

    /** 'False' value of a 'Bool' attribute. */
    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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception BadParameterException
     *      is thrown when the specified value does not conform to the attribute type.
     * exception DoesNotExistException
     *      is thrown when the attribute does not exist.
     * exception IncorrectStateException
     *      is thrown when the attribute is a vector attribute.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void setAttribute(String key, String value)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            IncorrectStateException, BadParameterException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Gets the value of an attribute.
     *
     * param key
     *      the attribute key.
     * return
     *      the value of this attribute.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception DoesNotExistException
     *      is thrown when the attribute does not exist.
     * exception IncorrectStateException
     *      is thrown when the attribute is a vector attribute.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public String getAttribute(String key) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Sets an attribute to an array of values.
     *
     * param key
     *      the attribute key.
     * param values
     *      values to set the attribute to.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception BadParameterException
     *      is thrown when the specified value does not conform to the attribute type.
     *      In particular, this exception is thrown when the specified value is
     *      <code>null</code>. An array with no elements is allowed, though.
     * exception DoesNotExistException
     *      is thrown when the attribute does not exist.
     * exception IncorrectStateException
     *      is thrown when the attribute is not a vector attribute.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void setVectorAttribute(String key, String[] values)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            IncorrectStateException, BadParameterException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Gets the array of values associated with an attribute.
     *
     * param key
     *      the attribute key.
     * return the values of this attribute, which may be an array with no
     *      elements, but not <code>null</code>.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception DoesNotExistException
     *      is thrown when the attribute does not exist.
     * exception IncorrectStateException
     *      is thrown when the attribute is not a vector attribute.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public String[] getVectorAttribute(String key)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            IncorrectStateException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * Removes an attribute.
     *
     * param key
     *      the attribute key.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception DoesNotExistException
     *      is thrown when the attribute does not exist.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void removeAttribute(String key) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * Gets the list of attribute keys.
     *
     * return
     *      the list of attribute keys.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public String[] listAttributes() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, TimeoutException, NoSuccessException;

    /**
     * Finds matching attributes.
     * This method accepts a list of patterns, and returns a list of keys for
     * those attributes that match any one of the specified patterns. The allowed
     * patterns describe both attribute keys and values, and are formatted as
     * <code>key-pattern=value-pattern</code>. Both the <code>key-pattern</code>
     * <code>value-pattern</code> can contain wildcards. The <code>value-pattern</code>
     * can be empty, and the method will then return all attribute keys which match the
     * <code>key-pattern</code>. The equal sign <code>=</code> can then be omitted
     * from the pattern.
     *
     * param patterns
     *      the search patterns.
     * return
     *      the list of matching attribute keys.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception BadParameterException
     *      is thrown when one ore more of the specified patterns are not correctly
     *      formatted.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public String[] findAttributes(String... patterns)
            throws NotImplementedException, BadParameterException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, TimeoutException, NoSuccessException;

    /**
     * Checks the existence of an attribute.
     *
     * param key
     *      the attribute key.
     * return
     *      <code>true</code> if the attribute exists.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public boolean existsAttribute(String key)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            TimeoutException, NoSuccessException;

    /**
     * Checks the attribute for being read-only.
     *
     * param key
     *      the attribute key.
     * return
     *      <code>true</code> if the attribute exists and is read-only.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception DoesNotExistException
     *      is thrown when the attribute does not exist.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public boolean isReadOnlyAttribute(String key)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Checks the attribute for being writable.
     *
     * param key
     *      the attribute key.
     * return
     *      <code>true</code> if the attribute exists and is writable.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception DoesNotExistException
     *      is thrown when the attribute does not exist.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public boolean isWritableAttribute(String key)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Checks the attribute for being removable.
     *
     * param key
     *      the attribute key.
     * return
     *      <code>true</code> if the attribute exists and is removable.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception DoesNotExistException
     *      is thrown when the attribute does not exist.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public boolean isRemovableAttribute(String key)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Checks the attribute for being a vector.
     *
     * param key
     *      the attribute key.
     * return
     *      <code>true</code> if the attribute is a vector attribute.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception DoesNotExistException
     *      is thrown when the attribute does not exist.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public boolean isVectorAttribute(String key)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            DoesNotExistException, TimeoutException, NoSuccessException;
}
```

3.8.2 AsyncAttributes

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

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

/**
 * Task versions of all methods from the <code>Attributes</code> interface.
 * The generic type <code>T</code> specifies the object type that implements
 * this interface.
 */
public interface AsyncAttributes<T> 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<T, Void> setAttribute(TaskMode mode, String key, String value)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<T, String> getAttribute(TaskMode mode, String key)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<T, Void> setVectorAttribute(TaskMode mode, String key,
            String[] values) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<T, String[]> getVectorAttribute(TaskMode mode, String key)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<T, Void> removeAttribute(TaskMode mode, String key)
            throws NotImplementedException;

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

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<T, String[]> findAttributes(TaskMode mode, String... patterns)
            throws NotImplementedException;

    /**
```

```
     * Creates a task that tests for the existence of an attribute.
     *
     * param mode
     *           determines the initial state of the task.
     * param key
     *           the attribute key.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<T, Boolean> existsAttribute(TaskMode mode, String key)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<T, Boolean> isReadOnlyAttribute(TaskMode mode, String key)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<T, Boolean> isWritableAttribute(TaskMode mode, String key)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<T, Boolean> isRemovableAttribute(TaskMode mode, String key)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<T, Boolean> isVectorAttribute(TaskMode mode, String key)
            throws NotImplementedException;
}
```

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.

Interfaces that extend `Async` and also should extend `Monitorable`, should extend `AsyncMonitorable` instead, as that provides the task versions of the methods in `Monitorable` as well. Likewise, interfaces that extend `Async` and also should extend `Steerable`, should extend `AsyncSteerable` instead, as that provides the task versions of the methods in `Steerable` as well.

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.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.IncorrectStateException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;

/**
 * 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.
     * param cb
     *      the callback to add.
     * return
     *      the cookie that identifies the callback in the metric.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown if the metric is "Final".
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public int addCallback(Callback cb) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * Removes a callback from the metric.
     *
     * param cookie
     *      the cookie that identifies the metric.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified cookie does not refer to an
     *      installed callback.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void removeCallback(int cookie) throws NotImplementedException,
            BadParameterException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            TimeoutException, NoSuccessException;

    /**
     * Pushes a new metric value to the backend.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the metric is "Final".
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void fire() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;
}
```

3.9.2 Callback

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

import org.ogf.saga.context.Context;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.NotImplementedException;

/**
 * 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
     *      whether the callback stays registered.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception AuthorizationFailedException
     *      is thrown when the cb method decides not to authorize this
     *      particular invocation.
     */
    public boolean cb(Monitorable mt, Metric metric, Context ctx)
            throws NotImplementedException, AuthorizationFailedException;
}
```

3.9.3 Monitorable

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

import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.DoesNotExistException;
import org.ogf.saga.error.IncorrectStateException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;

/**
 * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public String[] listMetrics() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, TimeoutException, NoSuccessException;

    /**
     * Returns a metric instance, identified by name.
     *
     * param name
     *      the name of the metric to be returned.
     * return
     *      the metric.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception DoesNotExistException
     *      is thrown when a metric with the specified name does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public Metric getMetric(String name) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception DoesNotExistException
     *      is thrown when a metric with the specified name does not exist.
     * exception IncorrectStateException
     *      is thrown if the specified metric has mode "Final".
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public int addCallback(String name, Callback cb)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            DoesNotExistException, TimeoutException, NoSuccessException,
            IncorrectStateException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception DoesNotExistException
     *      is thrown when a metric with the specified name does not exist.
     * exception BadParameterException
     *      is thrown when the specified cookie does not refer to a callback.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void removeCallback(String name, int cookie)
            throws NotImplementedException, DoesNotExistException,
            BadParameterException, TimeoutException, NoSuccessException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException;
}
```

3.9.4 AsyncMonitorable

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

import org.ogf.saga.error.NotImplementedException;
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<T> extends Monitorable {

    /**
     * Creates a task that lists all metrics associated with the object.
     *
     * param mode
     *           the task mode.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<T, String[]> listMetrics(TaskMode mode)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<T, Metric> getMetric(TaskMode mode, String name)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<T, Integer> addCallback(TaskMode mode, String name, Callback cb)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<T, Void> removeCallback(TaskMode mode, String name, int cookie)
            throws NotImplementedException;
}
```

3.9.5 Steerable

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

import org.ogf.saga.error.AlreadyExistsException;
import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.DoesNotExistException;
import org.ogf.saga.error.IncorrectStateException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;

/**
 * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception AlreadyExistsException
     *      is thrown when a metric with the same name already is already
     *      known for this object.
     * exception IncorrectStateException
     *      is thrown if the object does not support the addition of new
     *      metrics.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public boolean addMetric(Metric metric) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, AlreadyExistsException,
            TimeoutException, NoSuccessException, IncorrectStateException;

    /**
     * Removes a metric instance.
     *
     * param name
     *      the name of the metric to be removed.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception DoesNotExistException
     *      is thrown when a metric with the this name is not known
     *      to this object.
     * exception IncorrectStateException
     *      is thrown if the object does not support the removal of
     *      metrics.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void removeMetric(String name) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, DoesNotExistException, TimeoutException,
            NoSuccessException, IncorrectStateException;

    /**
     * Pushes a new metric value to the backend.
     *
     * param name
     *      the name of the metric to be fired.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception DoesNotExistException
     *      is thrown when a metric with the this name is not known
     *      to this object.
     * exception IncorrectStateException
     *      is thrown when an attempt is made to fire a "Final" or "ReadOnly"
     *      metric.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void fireMetric(String name) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            DoesNotExistException, TimeoutException, NoSuccessException;
}
```

3.9.6 AsyncSteerable

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

import org.ogf.saga.error.NotImplementedException;
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<T> 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<T, Boolean> addMetric(TaskMode mode, Metric metric)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<T, Void> removeMetric(TaskMode mode, String name)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<T, Void> fireMetric(TaskMode mode, String name)
            throws NotImplementedException;
}
```

3.9.7 MonitoringFactory

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

import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.TimeoutException;

/**
 * Factory for objects in the monitoring package.
 */
public abstract class MonitoringFactory {
    private static MonitoringFactory getFactory(String sagaFactoryName)
         throws NoSuccessException, NotImplementedException {
return ImplementationBootstrapLoader.getMonitoringFactory(sagaFactoryName);
    }

    /**
     * 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 NotImplementedException, BadParameterException,
            TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown on incorrectly formatted 'value' parameter, invalid 'mode'
     *      or 'type' parameter, and empty required parameter (all but 'unit').
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public synchronized static Metric createMetric(String name, String desc,
            String mode, String unit, String type, String value)
            throws NotImplementedException, BadParameterException,
            TimeoutException, NoSuccessException {
        return createMetric(null, name, desc, mode, unit, type, value);
    }

    /**
     * Constructs a <code>Metric</code> object with the specified parameters.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown on incorrectly formatted 'value' parameter, invalid 'mode'
     *      or 'type' parameter, and empty required parameter (all but 'unit').
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public synchronized static Metric createMetric(String sagaFactoryClassname, String name, String de
            String mode, String unit, String type, String value)
            throws NotImplementedException, BadParameterException,
            TimeoutException, NoSuccessException {
        return getFactory(sagaFactoryClassname).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.

Generic type parameters for Tasks A task has two generic type parameters: the type of the object that created the task, and the type of the return value of the task. Tasks can be stored in a `TaskContainer`. However, since a task container should be able to contain any task (and even jobs), the generic type information gets lost. So, when a task is pulled from a task container, the return type of `Task.getResult()` is `java.lang.Object`. If necessary, the user can associate information about the task with the task's UUID, and use that information to cast the result to the correct type. This is beyond the scope of Java generics.

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.AlreadyExistsException;
import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.DoesNotExistException;
import org.ogf.saga.error.IncorrectStateException;
import org.ogf.saga.error.IncorrectURLException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.SagaIOException;
import org.ogf.saga.error.TimeoutException;
import org.ogf.saga.monitoring.Monitorable;

/**
 * Tasks can only be created through asynchronous method calls. The generic
 * parameter <code>T</code> denotes the type of the object that generated this
 * task. Note that <code>T</code> cannot be specified as
 * <code>T extends SagaObject</code> because factories can also generate tasks
 * and factories are not Saga objects. The generic parameter <code>E</code>
 * denotes the type of the return value of the asynchronous method call.
 */
public interface Task<T, 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the task is not in NEW state.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void run() throws NotImplementedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the task is in NEW state.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void waitFor() throws NotImplementedException,
            IncorrectStateException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out. Note that this is not thrown when the specified timeout expires.
     *      In that case, <code>false</code> is returned.
     * exception IncorrectStateException
     *      is thrown when the task is in NEW state.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public boolean waitFor(float timeoutInSeconds)
            throws NotImplementedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the task is in NEW state.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void cancel() throws NotImplementedException,
            IncorrectStateException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the task is in NEW state.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void cancel(float timeoutInSeconds) throws NotImplementedException,
            IncorrectStateException, TimeoutException, NoSuccessException;

    /**
     * Gets the state of the task.
     *
     * return
     *      the state of the task.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public State getState() throws NotImplementedException, TimeoutException,
            NoSuccessException;

    /**
     * Obtains the result of the asynchronous method call. Implies a
     * {link #waitFor()}.
     * If the task is in {link State#FAILED} state after a
     * {link #waitFor()}, a {@link #rethrow()} is called.
     *
     * return
     *      the result.
     * exception IncorrectURLException
     *      is thrown when the failed task threw it.
     * exception NoSuccessException
     *      is thrown when the failed task threw it.
     * exception BadParameterException
     *      is thrown when the failed task threw it.
     * exception IncorrectStateException
     *      is thrown when the failed task threw it, or when the task is in
     *      {link State#NEW} or {@link State#CANCELED} state.
     * exception PermissionDeniedException
     *      is thrown when the failed task threw it.
     * exception AuthorizationFailedException
     *      is thrown when the failed task threw it.
     * exception AuthenticationFailedException
     *      is thrown when the failed task threw it.
     * exception NotImplementedException
     *      is thrown when the failed task threw it, or when
     *      {link #getResult()} is not implemented.
     * exception AlreadyExistsException
     *      is thrown when the failed task threw it.
     * exception DoesNotExistException
     *      is thrown when the failed task threw it.
     * exception TimeoutException
     *      is thrown when the failed task threw it.
     * exception SagaIOException
     *      is thrown when the failed task threw it.
     */
    public E getResult() throws NotImplementedException, IncorrectURLException,
            BadParameterException, AlreadyExistsException, DoesNotExistException,
            IncorrectStateException, PermissionDeniedException,
            AuthorizationFailedException, AuthenticationFailedException,
            TimeoutException, SagaIOException, NoSuccessException;

    /**
     * Gets the object from which the task was created.
     *
     * return
     *      the object this task was created from.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public T getObject() throws NotImplementedException, TimeoutException,
            NoSuccessException;

    /**
     * Throws any exception a failed task caught. It does nothing unless the
     * task is in state {link State#FAILED}. Rethrow can be called multiple times,
     * always throwing the same exception.
     *
     * exception IncorrectURLException
     *      is thrown when the failed task threw it.
     * exception NoSuccessException
     *      is thrown when the failed task threw it.
     * exception BadParameterException
     *      is thrown when the failed task threw it.
     * exception IncorrectStateException
     *      is thrown when the failed task threw it.
     * exception PermissionDeniedException
     *      is thrown when the failed task threw it.
     * exception AuthorizationFailedException
     *      is thrown when the failed task threw it.
     * exception AuthenticationFailedException
     *      is thrown when the failed task threw it.
     * exception NotImplementedException
     *      is thrown when the failed task threw it.
     * exception AlreadyExistsException
     *      is thrown when the failed task threw it.
     * exception DoesNotExistException
     *      is thrown when the failed task threw it.
     * exception TimeoutException
     *      is thrown when the failed task threw it.
     * exception SagaIOException
     *      is thrown when the failed task threw it.
     */
    public void rethrow() throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException, SagaIOException;
}
```

3.10.3 TaskContainer

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

import org.ogf.saga.SagaObject;
import org.ogf.saga.error.DoesNotExistException;
import org.ogf.saga.error.IncorrectStateException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.TimeoutException;
import org.ogf.saga.monitoring.Monitorable;

/**
 * Container object for tasks.
 * Note that a task container must be able to contain any task, so it has
 * no generic parameters for task result type or task object type. The
 * consequence of this is that these types are unknown. So, a task
 * pulled from a container has type <code>Task&lt;?,?&gt;</code>, and its
 * {link Task#getObject() getObject()} an {@link Task#getResult()}
 * methods return a {link java.lang.Object}.
 */
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 object identifier.
     */
    public static final String TASKCONTAINER_STATE = "task_container.state";

    /**
     * Adds a task to the task container. If the container already contains the
     * task, the method returns silently.
     *
     * param task
     *      the task to add.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void add(Task<?,?> task) throws NotImplementedException, TimeoutException,
            NoSuccessException;

    /**
     * Removes the specified task from this container.
     *
     * param task
     *      the task to be removed.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception DoesNotExistException
     *      is thrown when the container does not contain the specified task.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void remove(Task<?,?> task) throws NotImplementedException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Starts all asynchronous operations in the container.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when any of the tasks in the container
     *      is not in NEW state.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception DoesNotExistException
     *      is thrown when the container does not contain any tasks.
     */
    public void run() throws NotImplementedException, IncorrectStateException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Waits for all tasks to end up in a final state. This
     * method blocks indefinitely. One of the finished tasks is returned, and
     * removed from the task container.
     *
     * return
     *      any of the finished tasks.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      may be thrown when any of the tasks in the container
     *      would throw this on {link Task#waitFor()}.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception DoesNotExistException
     *      is thrown when the container does not contain any tasks.
     */
    public Task<?,?> waitFor() throws NotImplementedException,
            IncorrectStateException, DoesNotExistException, TimeoutException,
            NoSuccessException;


    /**
     * Waits for one or more of the tasks to end up in a final state. This
     * method blocks indefinitely. 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      may be thrown when any of the tasks in the container
     *      would throw this on {link Task#waitFor()}.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception DoesNotExistException
     *      is thrown when the container does not contain any tasks.
     */
    public Task<?,?> waitFor(WaitMode mode) throws NotImplementedException,
            IncorrectStateException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * Waits for all 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.
     * return
     *      any of the finished tasks.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      may be thrown when any of the tasks in the container
     *      would throw this on {link Task#waitFor()}.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception DoesNotExistException
     *      is thrown when the container does not contain any tasks.
     */
    public Task<?,?> waitFor(float timeoutInSeconds)
            throws NotImplementedException, IncorrectStateException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * 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
     *      any of the finished tasks.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      may be thrown when any of the tasks in the container
     *      would throw this on {link Task#waitFor()}.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception DoesNotExistException
     *      is thrown when the container does not contain any tasks.
     */
    public Task<?,?> waitFor(float timeoutInSeconds, WaitMode mode)
            throws NotImplementedException, IncorrectStateException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when any of the tasks in the container
     *      would throw this on {link Task#cancel()}.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception DoesNotExistException
     *      is thrown when the container does not contain any tasks.
     */
    public void cancel() throws NotImplementedException,
            IncorrectStateException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when any of the tasks in the container
     *      would throw this on {link Task#cancel()}.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception DoesNotExistException
     *      is thrown when the container does not contain any tasks.
     */
    public void cancel(float timeoutInSeconds) throws NotImplementedException,
            IncorrectStateException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * Returns the number of tasks in this task container.
     *
     * return the number of tasks.
     */
    public int size() throws NotImplementedException, TimeoutException,
            NoSuccessException;

    /**
     * Gets a single task from the task container.
     *
     * param id
     *      the object identifier of the task (typically obtained from the
     *      <code>TASKCONTAINER_STATE</code> metric).
     * return
     *      the task.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception DoesNotExistException
     *      is thrown when the container does not contain a task for the
     *      specified id.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public Task<?,?> getTask(String id) throws NotImplementedException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Gets the tasks in this task container.
     *
     * return
     *      the tasks.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public Task<?,?>[] getTasks() throws NotImplementedException, TimeoutException,
            NoSuccessException;
    /**
     * Gets the states of all tasks in the task container.
     *
     * return
     *      the states.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public State[] getStates() throws NotImplementedException,
            TimeoutException, NoSuccessException;
}
```

3.10.4 Taskfactory

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

import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.TimeoutException;

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

    private static TaskFactory getFactory(String sagaFactoryName)
         throws NoSuccessException, NotImplementedException {
return ImplementationBootstrapLoader.getTaskFactory(sagaFactoryName);
    }

    /**
     * Constructs a <code>TaskContainer</code> object. This method is to be
     * provided by the factory.
     *
     * return the task container.
     */
    protected abstract TaskContainer doCreateTaskContainer()
            throws NotImplementedException, TimeoutException,
            NoSuccessException;
    /**
     * Constructs a <code>TaskContainer</code> object. This method is to be
     * provided by the factory.
     *
     * return
     *      the task container.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public synchronized static TaskContainer createTaskContainer()
            throws NotImplementedException, TimeoutException,
            NoSuccessException {
        return createTaskContainer(null);
    }

    /**
     * Constructs a <code>TaskContainer</code> object. This method is to be
     * provided by the factory.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * return
     *      the task container.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public synchronized static TaskContainer createTaskContainer(String sagaFactoryClassname)
            throws NotImplementedException, TimeoutException,
            NoSuccessException {
        return getFactory(sagaFactoryClassname).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 {

    /** Describes a newly constructed task instance which has not yet run. */
    NEW(1),
    /**
     * The {link Task#run()} method has been invoked on the task, either
     * explicitly or implicitly.
     */
    RUNNING(2),
    /** The task has finished successfully. This state is final. */
    DONE(3),
    /**
     * The task has been cancelled, that is, {link Task#cancel()} has been called on it.
     * This state is final.
     */
    CANCELED(4),
    /** The task has finished unsuccessfully. This state is final. */
    FAILED(5),
    /**
     * This is actually a job state, but is included here because enumerations
     * cannot be extended. It indicates that the job instance has been suspended.
     */
    SUSPENDED(6);

    private int value;

    private 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 {

    /**
     * The task is started in {link State#RUNNING} state, that is, the {@link Task#run()} call
     * is implicit.
     */
    ASYNC(1),
    /** The task is started in {link State#NEW} state. */
    TASK(2),
    /** The task is started in {link State#RUNNING} state, and is waited for. */
    SYNC(3);

    private int value;

    private 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 {
    /**
```

```
     *  Wait for all tasks in the container. {link Task#waitFor()} only returns
     *  when all tasks in the container have reached a final state.
     */
    ALL(0),
    /**
     *  Wait for any task in the container. {link Task#waitFor()} when one or more
     * tasks in the container have reached a final state.
     */
    ANY(1);

    private int value;

    private 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`, `AuthorizationFailed`, `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.

An important deviation from the language-independent SAGA specifications is that the `JobService.runJob` method is specified differently: 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 `Job.getStdin`, `Job.getStdout`, and `Job.getStderr` methods.

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.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.DoesNotExistException;
import org.ogf.saga.error.IncorrectStateException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;
import org.ogf.saga.permissions.Permissions;
import org.ogf.saga.task.Async;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;

/**
 * Jobs are created by a {link JobService}, using a {@link JobDescription}.
 * The <code>Job</code> interface extends the
 * {link org.ogf.saga.task.Task Task} interface, but jobs don't have an
 * associated Saga object or a result object, so the generic instantiation
 * parameters are {link java.lang.Void Void}.
 */
public interface Job extends Task<Void, Void>, Async, AsyncAttributes<Job>,
        Permissions<Job> {

    // Required attributes:

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

    /** Attribute name, URL representation of the JobService that created the job. */
```

```
public static final String SERVICEURL = "ServiceURL";

// 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
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception DoesNotExistException
     *      is thrown in cases where the job description is not available,
     *      for instance when the job was not submitted through SAGA and the
     *      job was obtained using the {link JobService#getJob(String)} call.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public JobDescription getJobDescription() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * Returns the input stream of this job (to which can be written).
     *
     * return
     *      the stream.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception DoesNotExistException
     *      is thrown when the stream is not available for some reason.
     * exception IncorrectStateException
     *      is thrown when the job is not interactive.
     * exception BadParameterException
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public OutputStream getStdin() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, TimeoutException, IncorrectStateException,
            NoSuccessException;

    /**
     * Returns the output stream of this job (which can be read).
     *
     * return
     *      the stream.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception DoesNotExistException
     *      is thrown when the stream is not available for some reason.
     * exception IncorrectStateException
     *      is thrown when the job is not interactive.
     * exception BadParameterException
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public InputStream getStdout() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, TimeoutException, IncorrectStateException,
            NoSuccessException;

    /**
     * Returns the error stream of this job (which can be read).
     *
     * return
     *      the stream.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception DoesNotExistException
     *      is thrown when the stream is not available for some reason.
     * exception IncorrectStateException
     *      is thrown when the job is not interactive.
     * exception BadParameterException
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public InputStream getStderr() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, TimeoutException, IncorrectStateException,
            NoSuccessException;

    /**
     * Asks the resource manager to perform a suspend operation on a running
     * job.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the job is not in RUNNING state.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void suspend() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
```

```
     * Asks the resource manager to perform a resume operation on a suspended
     * job.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the job is not in SUSPENDED state.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void resume() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * Asks the resource manager to initiate a checkpoint operation on a running
     * job.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the job is not in RUNNING state.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void checkpoint() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * Asks the resource manager to migrate a job.
     *
     * param jd
     *      new job parameters to apply when the job is migrated.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the job is not in RUNNING or SUSPENDED state.
     * exception BadParameterException
     *      is thrown if there is something wrong with the specified
     *      job description.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void migrate(JobDescription jd) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException;

    /**
     * Asks the resource manager to deliver an arbitrary signal to a dispatched
     * job.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the job is not in RUNNING or SUSPENDED state.
     * exception BadParameterException
     *      is thrown if the specified signal is not supported by the backend.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void signal(int signum) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException;

    //
    // 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<Job, JobDescription> getJobDescription(TaskMode mode)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<Job, OutputStream> getStdin(TaskMode mode)
            throws NotImplementedException;

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

    /**
     * Creates a task that obtains the error stream of this job (which can be
     * read).
     *
     * param mode
     *           the task mode.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<Job, InputStream> getStderr(TaskMode mode)
```

```
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<Job, Void> suspend(TaskMode mode)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<Job, Void> resume(TaskMode mode) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<Job, Void> checkpoint(TaskMode mode)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<Job, Void> migrate(TaskMode mode, JobDescription jd)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<Job, Void> signal(TaskMode mode, int signum)
            throws NotImplementedException;
}
```

4.1.2 JobDescription

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

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

/**
 * The contents of a job description is defined by its 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, hard limit for the total job runtime. */
    public static final String WALLTIMELIMIT = "WallTimeLimit";

    /**
     * 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, name of an account or project name. */
    public static final String JOBPROJECT = "JobProject";

    /**
     * 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<JobSelf> {
}
```

4.1.4 Jobservice

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

import java.util.List;
import org.ogf.saga.SagaObject;
import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.DoesNotExistException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;
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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception BadParameterException
     *      is thrown when the job description does not contain a valid
     *      EXECUTABLE, or contains invalid values.
     */
    public Job createJob(JobDescription jd) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException, TimeoutException,
            NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception BadParameterException
     *      is thrown when the command line cannot be parsed.
     */
    public Job runJob(String commandLine, String host, boolean interactive)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception BadParameterException
     *      is thrown when the command line cannot be parsed.
     */
    public Job runJob(String commandLine, String host)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception BadParameterException
     *      is thrown when the command line cannot be parsed.
     */
    public Job runJob(String commandLine, boolean interactive)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception BadParameterException
     *      is thrown when the command line cannot be parsed.
     */
    public Job runJob(String commandLine) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException, TimeoutException,
            NoSuccessException;

    /**
     * Obtains the list of jobs that are currently known to the resource
     * manager.
     *
     * return
     *      a list of job identifications.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public List<String> list() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, TimeoutException, NoSuccessException;

    /**
     * Returns the job instance associated with the specified job
     * identification.
     *
     * param jobId
     *      the job identification.
     * return
     *      the job instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception BadParameterException
     *      is thrown when the JobService cannot parse the job id.
     * exception DoesNotExistException
     *      is thrown when the JobService can handle the job id, but the
     *      referenced job is not alive.
     */
    public Job getJob(String jobId) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Returns a job instance representing the calling application.
     *
     * return
     *      the job instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public JobSelf getSelf() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, TimeoutException, NoSuccessException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<JobService, Job> createJob(TaskMode mode, JobDescription jd)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<JobService, List<String>> list(TaskMode mode)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<JobService, Job> getJob(TaskMode mode, String jobId)
```

```
            throws NotImplementedException;

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

    /**
     * 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<JobService, Job> runJob(TaskMode mode, String commandLine,
            String host, boolean interactive) throws NotImplementedException;

    /**
     * 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<JobService, Job> runJob(TaskMode mode, String commandLine,
            String host) throws NotImplementedException;

    /**
     * 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<JobService, Job> runJob(TaskMode mode, String commandLine,
            boolean interactive) throws NotImplementedException;

    /**
     * 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<JobService, Job> runJob(TaskMode mode, String commandLine)
            throws NotImplementedException;
}
```

4.1.5 JobFactory

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

import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.IncorrectURLException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;
import org.ogf.saga.session.Session;
import org.ogf.saga.session.SessionFactory;
import org.ogf.saga.task.TaskMode;
import org.ogf.saga.task.Task;
import org.ogf.saga.url.URL;
import org.ogf.saga.url.URLFactory;

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

    private static JobFactory getFactory(String sagaFactoryName)
            throws NoSuccessException, NotImplementedException {
return ImplementationBootstrapLoader.getJobFactory(sagaFactoryName);
    }

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

    /**
     * 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 NotImplementedException, BadParameterException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, TimeoutException, NoSuccessException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    protected abstract Task<JobFactory, JobService> doCreateJobService(
            TaskMode mode, Session session, URL rm)
            throws NotImplementedException;

    /**
     * Creates a job description.
     *
     * return
     *      the job description.
     * exception NotImplementedException
     *      is thrown when this method is not implemented.
     * throws NoSuccessException
     *      is thrown when the Saga factory could not be created.
     */
    public static JobDescription createJobDescription()
            throws NotImplementedException, NoSuccessException {
        return createJobDescription(null);
    }

    /**
     * Creates a job description.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * return
     *      the job description.
     * exception NotImplementedException
     *      is thrown when this method is not implemented.
     * throws NoSuccessException
     *      is thrown when the Saga factory could not be created.
     */
    public static JobDescription createJobDescription(String sagaFactoryClassname)
```

```
            throws NotImplementedException, NoSuccessException {
        return getFactory(sagaFactoryClassname).doCreateJobDescription();
    }

    /**
     * Creates a job service.
     *
     * param session
     *      the session handle.
     * param rm
     *      contact string for the resource manager.
     * return
     *      the job service.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception BadParameterException
     *      is thrown if the specified URL cannot be contacted, or a
     *      default contact point does not exist or cannot be found.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static JobService createJobService(Session session, URL rm)
            throws NotImplementedException, BadParameterException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, TimeoutException, NoSuccessException {
        return createJobService((String)null, session, rm);
    }
    /**
     * Creates a job service.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param session
     *      the session handle.
     * param rm
     *      contact string for the resource manager.
     * return
     *      the job service.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception BadParameterException
     *      is thrown if the specified URL cannot be contacted, or a
     *      default contact point does not exist or cannot be found.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static JobService createJobService(String sagaFactoryClassname, Session session, URL rm)
            throws NotImplementedException, BadParameterException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, TimeoutException, NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
if (rm == null) {
    rm = URLFactory.createURL(sagaFactoryClassname, "");
}
        return getFactory(sagaFactoryClassname).doCreateJobService(session, rm);
    }


    /**
     * Creates a job service using the default contact string.
     *
     * param session
     *      the session handle.
     * return
     *      the job service.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception BadParameterException
     *      is thrown if a default contact point does not exist or cannot be found.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static JobService createJobService(Session session)
            throws NotImplementedException, BadParameterException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, TimeoutException, NoSuccessException {
        return createJobService(session, (URL) null);
    }


    /**
     * Creates a job service using the default contact string.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param session
     *      the session handle.
     * return
     *      the job service.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception BadParameterException
     *      is thrown if a default contact point does not exist or cannot be found.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static JobService createJobService(String sagaFactoryClassname, Session session)
            throws NotImplementedException, BadParameterException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, TimeoutException, NoSuccessException {
        return createJobService(sagaFactoryClassname, session, (URL) null);
    }
    /**
     * Creates a job service, using the default session.
     *
     * param rm
     *      contact string for the resource manager.
     * return
     *      the job service.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception BadParameterException
     *      is thrown if the specified URL cannot be contacted, or a
     *      default contact point does not exist or cannot be found.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static JobService createJobService(URL rm)
            throws NotImplementedException, BadParameterException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, TimeoutException, NoSuccessException {
        return createJobService((Session) null, rm);
    }
    /**
     * Creates a job service, using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param rm
     *      contact string for the resource manager.
     * return
     *      the job service.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception BadParameterException
     *      is thrown if the specified URL cannot be contacted, or a
     *      default contact point does not exist or cannot be found.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static JobService createJobService(String sagaFactoryClassname, URL rm)
            throws NotImplementedException, BadParameterException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, TimeoutException, NoSuccessException {
        return createJobService(sagaFactoryClassname, (Session) null, rm);
    }


    /**
     * Creates a job service, using the default session and default contact
     * string.
     *
     * return
     *      the job service.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception BadParameterException
     *      is thrown if a default contact point does not exist or cannot be found.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static JobService createJobService() throws NotImplementedException,
            BadParameterException, IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            TimeoutException, NoSuccessException {
        return createJobService((Session) null);
    }


    /**
     * Creates a job service, using the default session and default contact
     * string.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * return
     *      the job service.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception BadParameterException
     *      is thrown if a default contact point does not exist or cannot be found.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static JobService createJobService(String sagaFactoryClassname) throws NotImplementedExcept
            BadParameterException, IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            TimeoutException, NoSuccessException {
        return createJobService(sagaFactoryClassname, (Session) null);
    }


    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<JobFactory, JobService> createJobService(TaskMode mode,
            Session session, URL rm) throws NotImplementedException,
            NoSuccessException {
        return createJobService((String) null, mode, session, rm);
    }

    /**
     * Creates a task that creates a job service.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param session
     *            the session handle.
     * param rm
     *            contact string for the resource manager.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<JobFactory, JobService> createJobService(String sagaFactoryClassname, TaskMode
            Session session, URL rm) throws NotImplementedException,
            NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
if (rm == null) {
    try {
rm = URLFactory.createURL(sagaFactoryClassname, "");
    } catch (BadParameterException e) {
// Should not happen
throw new NoSuccessException("Unexpected exception", e);
    }
}
        return getFactory(sagaFactoryClassname).doCreateJobService(mode, session, rm);
    }

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *               implemented.
     * throws NoSuccessException
     *            is thrown when the Saga factory could not be created.
     */
    public static Task<JobFactory, JobService> createJobService(TaskMode mode,
            Session session) throws NotImplementedException, NoSuccessException {
        return createJobService(mode, session, (URL) null);
    }


    /**
     * Creates a task that creates a job service, using a default contact
     * string.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param session
     *            the session handle.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<JobFactory, JobService> createJobService(String sagaFactoryClassname, TaskMode
            Session session) throws NotImplementedException, NoSuccessException {
        return createJobService(sagaFactoryClassname, mode, session, (URL) null);
    }

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the Saga factory could not be created or
     *                when the default session could not be created.
     */
    public static Task<JobFactory, JobService> createJobService(TaskMode mode,
            URL rm) throws NotImplementedException, NoSuccessException {
        return createJobService(mode, (Session) null, rm);
    }

    /**
     * Creates a task that creates a job service, using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param rm
     *            contact string for the resource manager.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the Saga factory could not be created or
     *                when the default session could not be created.
     */
    public static Task<JobFactory, JobService> createJobService(String sagaFactoryClassname, TaskMode
            URL rm) throws NotImplementedException, NoSuccessException {
        return createJobService(sagaFactoryClassname, mode, (Session) null, rm);
    }

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the Saga factory could not be created or
     *                when the default session could not be created.
     */
    public static Task<JobFactory, JobService> createJobService(TaskMode mode)
            throws NotImplementedException, NoSuccessException {
        return createJobService(mode, (URL) null);
    }


    /**
     * Creates a task that creates a job service, using the default session and
     * default contact string.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *           the task mode.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     * exception NoSuccessException
     *               is thrown when the Saga factory could not be created or
     *               when the default session could not be created.
     */
    public static Task<JobFactory, JobService> createJobService(String sagaFactoryClassname, TaskMode
            throws NotImplementedException, NoSuccessException {
        return createJobService(sagaFactoryClassname, mode, (URL) null);
    }
}
```

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.

The possible file flags are defined in the `Flags` enumeration class, which includes methods to combine them into integers, and to determine if they are set in an integer. These methods are not described in the language-independent SAGA specification, but are added in the Java language bindings because in Java, enumerations cannot be treated as integers. Also, since enumeration classes are not extensible in Java, flags from the `logicalfile` package and the `file` package are included here as well.

An `NSDirectory` represents a namespace entry that is a directory, and defines additional methods for them. A Java specific extension is that this interface extends `Iterable`, which allows an application to iterate over the entries in this directory, like this:

Code Example

```
r (URL url : directory) {
  // do something useful with url
  ...

```

4.2.1 NSDirectory

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

import java.util.List;

import org.ogf.saga.error.AlreadyExistsException;
import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.DoesNotExistException;
import org.ogf.saga.error.IncorrectStateException;
import org.ogf.saga.error.IncorrectURLException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;
import org.ogf.saga.url.URL;

/**
 * Represents a namespace entry that is a directory, and defines additional
 * methods for them. This interface extends <code>Iterable</code>, which
 * allows an application to iterate over the entries in this directory.
 * Implementations can use {link #getNumEntries()} and {@link #getEntry(int)}
 * to implement an iterator, but will have to encapsulate the exceptions that
 * these methods can throw in either a {link java.lang.RuntimeException} or a
 * {link java.lang.Error}.
 */
public interface NSDirectory extends NSEntry, Iterable<URL> {

    /**
     * Changes the working directory.
     *
     * param dir
     *      the directory to change to.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception BadParameterException
     *      is thrown when the specified URL contains an invalid directory
     *      name.
     * exception DoesNotExistException
     *      is thrown when the specified directory does not exist.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     */
    public void changeDir(URL dir) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Lists entries in the directory that match the specified pattern. If the
     * pattern is an empty string, all entries are listed. The only allowed flags
     * are NONE and DEREFERENCE.
     *
     * param pattern
     *      name or pattern to list.
     * param flags
     *      defining the operation modus.
     * return
     *      the matching entries.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when illegal flags are specified: only NONE and
     *      DEREFERENCE are allowed.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public List<URL> list(String pattern, int flags)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException, TimeoutException,
            NoSuccessException, IncorrectURLException;

    /**
     * Lists entries in the directory. The only allowed flag are NONE and DEREFERENCE.
     *
     * param flags
     *      defining the operation modus.
     * return
     *      the directory entries.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when illegal flags are specified: only NONE and
     *      DEREFERENCE are allowed.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public List<URL> list(int flags) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException,
            IncorrectURLException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when illegal flags are specified: only NONE and
     *      DEREFERENCE are allowed.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public List<URL> list(String pattern) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException,
            IncorrectURLException;

    /**
     * Lists entries in the directory.
     *
     * return
     *      the directory entries.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when illegal flags are specified: only NONE and
     *      DEREFERENCE are allowed.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public List<URL> list() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException,
            IncorrectURLException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when illegal flags are specified: only RECURSIVE is allowed.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public List<URL> find(String pattern, int flags)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException, TimeoutException,
            NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when illegal flags are specified: only RECURSIVE is allowed.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public List<URL> find(String pattern) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException;

    /**
     * Queries for the existence of an entry.
     *
     * param name
     *      to be tested for existence.
     * return
     *      <code>true</code> if the name exists.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL contains an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public boolean exists(URL name) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException, TimeoutException,
            NoSuccessException;

    /**
     * Returns the time of the last modification in seconds since epoch
     * (01.01.1970) of the specified name.
     *
     * param name
     *      the name of which the last modification time must be returned.
     * return
     *      the last modification time.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL contains an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception DoesNotExistException
     *      is thrown if the specified name does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public long getMTime(URL name) throws NotImplementedException,
            IncorrectURLException, DoesNotExistException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException, TimeoutException,
            NoSuccessException;

    /**
     * Tests the name for being a directory.
     *
     * param name
     *      to be tested.
     * return
     *      <code>true</code> if the name represents a directory.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL contains an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception DoesNotExistException
     *      is thrown if the specified name does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public boolean isDir(URL name) throws NotImplementedException,
            IncorrectURLException, DoesNotExistException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL contains an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception DoesNotExistException
     *      is thrown if the specified name does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public boolean isEntry(URL name) throws NotImplementedException,
            IncorrectURLException, DoesNotExistException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException;

    /**
     * Tests the name for being a link.
     *
     * param name
     *      to be tested.
     * return
     *      <code>true</code> if the name represents a link.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL contains an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception DoesNotExistException
     *      is thrown if the specified name does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public boolean isLink(URL name) throws NotImplementedException,
            IncorrectURLException, DoesNotExistException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException;

    /**
     * Returns the URL representing the link target.
     *
     * param name
     *      the name of the link.
     * return
     *      the resolved name.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL contains an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception DoesNotExistException
     *      is thrown if the specified name does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public URL readLink(URL name) throws NotImplementedException,
            IncorrectURLException, DoesNotExistException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException;

    /**
     * Obtains the number of entries in this directory.
     *
     * return
     *      the number of entries.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public int getNumEntries() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
```

```
     * Gives the name of an entry in the directory based upon the enumeration
     * defined by {link #getNumEntries()}.
     *
     * param entry
     *      index of the entry to get.
     * return
     *      the name of the entry.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception DoesNotExistException
     *      is thrown when the index is invalid.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public URL getEntry(int entry) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URLs contain an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified target URL already exists, and the
     *      <code>OVERWRITE</code> flag is not given.
     * exception DoesNotExistException
     *      is thrown if the source does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void copy(URL source, URL target, int flags)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            IncorrectURLException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Copies the source entry to another part of the namespace.
     *
     * param source
     *      name to copy.
     * param target
     *      name to copy to.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URLs contain an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified target URL already exists, and the
     *      <code>OVERWRITE</code> flag is not given.
     * exception DoesNotExistException
     *      is thrown if the source does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void copy(URL source, URL target) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectURLException,
            BadParameterException, IncorrectStateException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URLs contain an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified target URL already exists, and the
     *      <code>OVERWRITE</code> flag is not given.
     * exception DoesNotExistException
     *      is thrown if the source does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void copy(String source, URL target, int flags)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            IncorrectURLException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URLs contain an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified target URL already exists, and the
     *      <code>OVERWRITE</code> flag is not given.
     * exception DoesNotExistException
     *      is thrown if the source does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void copy(String source, URL target) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectURLException,
            BadParameterException, IncorrectStateException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;
    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URLs contain an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified target URL already exists, and the
     *      <code>OVERWRITE</code> flag is not given.
     * exception DoesNotExistException
     *      is thrown if the source does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void link(URL source, URL target, int flags)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            IncorrectURLException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Creates a symbolic link from the specified target to the specified
     * source.
     *
     * param source
     *      name to link to.
     * param target
     *      name of the link.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URLs contain an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified target URL already exists, and the
     *      <code>OVERWRITE</code> flag is not given.
     * exception DoesNotExistException
     *      is thrown if the source does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void link(URL source, URL target) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectURLException,
            BadParameterException, IncorrectStateException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URLs contain an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified target URL already exists, and the
     *      <code>OVERWRITE</code> flag is not given.
     * exception DoesNotExistException
     *      is thrown if the source does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void link(String source, URL target, int flags)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            IncorrectURLException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URLs contain an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified target URL already exists, and the
     *      <code>OVERWRITE</code> flag is not given.
     * exception DoesNotExistException
     *      is thrown if the source does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void link(String source, URL target) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectURLException,
            BadParameterException, IncorrectStateException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URLs contain an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified target URL already exists, and the
     *      <code>OVERWRITE</code> flag is not given.
     * exception DoesNotExistException
     *      is thrown if the source does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void move(URL source, URL target, int flags)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            IncorrectURLException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URLs contain an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified target URL already exists, and the
     *      <code>OVERWRITE</code> flag is not given.
     * exception DoesNotExistException
     *      is thrown if the source does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void move(URL source, URL target) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectURLException,
            BadParameterException, IncorrectStateException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URLs contain an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified target URL already exists, and the
     *      <code>OVERWRITE</code> flag is not given.
     * exception DoesNotExistException
     *      is thrown if the source does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void move(String source, URL target, int flags)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            IncorrectURLException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URLs contain an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified target URL already exists, and the
     *      <code>OVERWRITE</code> flag is not given.
     * exception DoesNotExistException
     *      is thrown if the source does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void move(String source, URL target) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectURLException,
            BadParameterException, IncorrectStateException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * Removes the specified entry.
     *
     * param target
     *      name to remove.
     * param flags
     *      defining the operation modus.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified name is an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception DoesNotExistException
     *      is thrown if the target does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void remove(URL target, int flags) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectURLException,
            BadParameterException, IncorrectStateException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Removes the specified entry.
     *
     * param target
     *      name to remove.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified name is an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception DoesNotExistException
     *      is thrown if the target does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void remove(URL target) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectURLException,
            BadParameterException, IncorrectStateException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Removes the specified entry. The target string may contain wildcards.
     *
     * param target
     *      name to remove.
     * param flags
     *      defining the operation modus.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified name is an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception DoesNotExistException
     *      is thrown if the target does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void remove(String target, int flags)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            IncorrectURLException, BadParameterException,
            IncorrectStateException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * Removes the specified entry. The target string may contain wildcards.
     *
     * param target
     *      name to remove.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified name is an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception DoesNotExistException
     *      is thrown if the target does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void remove(String target) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectURLException,
            BadParameterException, IncorrectStateException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Creates a new directory.
     *
     * param target
     *      directory to create.
     * param flags
     *      defining the operation modus.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>EXCLUSIVE</code> flag is given.
     * exception DoesNotExistException
     *      is thrown if the parent directory does not exist and the
     *      <code>CREATEPARENTS</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void makeDir(URL target, int flags) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * Creates a new directory.
     *
     * param target
     *      directory to create.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>EXCLUSIVE</code> flag is given.
     * exception DoesNotExistException
     *      is thrown if the parent directory does not exist and the
     *      <code>CREATEPARENTS</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void makeDir(URL target) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * Creates a new <code>NamespaceDirectory</code> instance.
     *
     * param name
     *      directory to open.
     * param flags
     *      defining the operation modus.
     * return
     *      the opened directory instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL does not point to a directory,
     *      or is an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public NSDirectory openDir(URL name, int flags)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Creates a new <code>NamespaceDirectory</code> instance.
     *
     * param name
     *      directory to open.
     * return
     *      the opened directory instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL does not point to a directory,
     *      or is an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      not thrown, but specified because a method may be invoked
     *      that can throw this exception, but will not in this case.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public NSDirectory openDir(URL name) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * Creates a new <code>NamespaceEntry</code> instance.
     *
     * param name
     *      entry to open.
     * param flags
     *      defining the operation modus.
     * return
     *      the opened entry instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL points to a directory,
     *      or is an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public NSEntry open(URL name, int flags) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * Creates a new <code>NamespaceEntry</code> instance.
     *
     * param name
     *      entry to open.
     * return
     *      the opened entry instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL points to a directory,
     *      or is an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      not thrown, but specified because a method may be invoked
     *      that can throw this exception, but will not in this case.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public NSEntry open(URL name) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the given id is unknown or not supported, or illegal
     *      flags are specified, or RECURSIVE is specified on a non-directory.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void permissionsAllow(URL target, String id, int permissions,
            int flags) throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            BadParameterException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the given id is unknown or not supported.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void permissionsAllow(URL target, String id, int permissions)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException, IncorrectURLException,
            IncorrectStateException, BadParameterException, TimeoutException,
            NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the given id is unknown or not supported, or illegal
     *      flags are specified, or RECURSIVE is specified on a non-directory.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void permissionsAllow(String target, String id, int permissions,
            int flags) throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            BadParameterException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the given id is unknown or not supported.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void permissionsAllow(String target, String id, int permissions)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException, IncorrectURLException,
            IncorrectStateException, BadParameterException, TimeoutException,
            NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the given id is unknown or not supported, or illegal
     *      flags are specified, or RECURSIVE is specified on a non-directory.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void permissionsDeny(URL target, String id, int permissions,
            int flags) throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException, TimeoutException,
            NoSuccessException, IncorrectStateException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the given id is unknown or not supported.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void permissionsDeny(URL target, String id, int permissions)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException, IncorrectURLException,
            BadParameterException, TimeoutException, NoSuccessException,
            IncorrectStateException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the given id is unknown or not supported, or illegal
     *      flags are specified, or RECURSIVE is specified on a non-directory.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void permissionsDeny(String target, String id, int permissions,
            int flags) throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException, TimeoutException,
            NoSuccessException, IncorrectStateException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the given id is unknown or not supported.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void permissionsDeny(String target, String id, int permissions)
            throws NotImplementedException, AuthenticationFailedException, IncorrectStateException,
            AuthorizationFailedException, PermissionDeniedException, IncorrectURLException,
            BadParameterException, TimeoutException, NoSuccessException;

    //
    // 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<NSDirectory, Void> changeDir(TaskMode mode, URL dir)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, List<URL>> list(TaskMode mode, String pattern,
            int flags) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, List<URL>> list(TaskMode mode, String pattern)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, List<URL>> list(TaskMode mode, int flags)
            throws NotImplementedException;

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

    /**
     * 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.
     * param pattern
     *           name or pattern to find.
     * param flags
     *           defining the operation modus.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<NSDirectory, List<URL>> find(TaskMode mode, String pattern,
            int flags) throws NotImplementedException;

    /**
     * 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.
     * param pattern
     *           name or pattern to find.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<NSDirectory, List<URL>> find(TaskMode mode, String pattern)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<NSDirectory, Boolean> exists(TaskMode mode, URL name)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<NSDirectory, Boolean> isDir(TaskMode mode, URL name)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<NSDirectory, Boolean> isEntry(TaskMode mode, URL name)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<NSDirectory, Boolean> isLink(TaskMode mode, URL name)
```

```
            throws NotImplementedException;

    /**
     * Creates a task that determines the last modification time of
     * the specified name.
     *
     * param mode
     *           the task mode.
     * param name
     *           of which the last modification time must be determined.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Long> getMTime(TaskMode mode, URL name)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, URL> readLink(TaskMode mode, URL name)
            throws NotImplementedException;

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

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, URL> getEntry(TaskMode mode, int entry)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> copy(TaskMode mode, URL source, URL target,
            int flags) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> copy(TaskMode mode, URL source, URL target)
            throws NotImplementedException;
```

```
/**
 * 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 NotImplementedException
 *               is thrown when the task version of this method is not
 *               implemented.
 */
public Task<NSDirectory, Void> copy(TaskMode mode, String source,
        URL target, int flags) throws NotImplementedException;

/**
 * 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 NotImplementedException
 *               is thrown when the task version of this method is not
 *               implemented.
 */
public Task<NSDirectory, Void> copy(TaskMode mode, String source, URL target)
        throws NotImplementedException;

/**
 * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> link(TaskMode mode, URL source, URL target,
            int flags) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> link(TaskMode mode, URL source, URL target)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> link(TaskMode mode, String source,
            URL target, int flags) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> link(TaskMode mode, String source, URL target)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> move(TaskMode mode, URL source, URL target,
            int flags) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> move(TaskMode mode, URL source, URL target)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> move(TaskMode mode, String source,
            URL target, int flags) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> move(TaskMode mode, String source, URL target)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> remove(TaskMode mode, URL target, int flags)
            throws NotImplementedException;

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

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> remove(TaskMode mode, String target,
            int flags) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> remove(TaskMode mode, String target)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> makeDir(TaskMode mode, URL target, int flags)
            throws NotImplementedException;

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

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, NSDirectory> openDir(TaskMode mode, URL name,
            int flags) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, NSDirectory> openDir(TaskMode mode, URL name)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, NSEntry> open(TaskMode mode, URL name, int flags)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<NSDirectory, NSEntry> open(TaskMode mode, URL name)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<NSDirectory, Void> permissionsAllow(TaskMode mode, URL target,
            String id, int permissions, int flags)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<NSDirectory, Void> permissionsAllow(TaskMode mode, URL target,
            String id, int permissions) throws NotImplementedException;
    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> permissionsAllow(TaskMode mode,
            String target, String id, int permissions, int flags)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> permissionsAllow(TaskMode mode,
            String target, String id, int permissions)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> permissionsDeny(TaskMode mode, URL target,
            String id, int permissions, int flags)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> permissionsDeny(TaskMode mode, URL target,
            String id, int permissions) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> permissionsDeny(TaskMode mode,
            String target, String id, int permissions, int flags)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSDirectory, Void> permissionsDeny(TaskMode mode,
            String target, String id, int permissions)
            throws NotImplementedException;
}
```

4.2.2 NSEntry

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

import org.ogf.saga.SagaObject;
import org.ogf.saga.error.AlreadyExistsException;
import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.DoesNotExistException;
import org.ogf.saga.error.IncorrectStateException;
import org.ogf.saga.error.IncorrectURLException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;
import org.ogf.saga.permissions.Permissions;
import org.ogf.saga.task.Async;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;
import org.ogf.saga.url.URL;

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

    /**
     * Obtains the complete URL referring to the entry.
     *
     * return
     *      the URL.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the NSEntry is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public URL getURL() throws NotImplementedException,
            IncorrectStateException, TimeoutException, NoSuccessException;

    /**
     * Obtains the current working directory for the entry.
     *
     * return
     *      the current working directory.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the NSEntry is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public URL getCWD() throws NotImplementedException,
            IncorrectStateException, TimeoutException, NoSuccessException;

    /**
     * Obtains the name part of the URL of this entry.
     *
     * return
     *      the name part.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the NSEntry is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public URL getName() throws NotImplementedException,
            IncorrectStateException, TimeoutException, NoSuccessException;

    /**
     * Tests this entry for being a directory.
     *
     * return
     *      true if the entry is a directory.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the NSEntry is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public boolean isDir() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the NSEntry is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public boolean isEntry() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * Tests this entry for being a link.
     *
     * return
     *      true if the entry is a link.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the NSEntry is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public boolean isLink() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * Returns the URL representing the link target. Resolves one link level
     * only.
     *
     * return
     *      the link target.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the NSEntry is already closed, or does not refer
     *      to a link.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public URL readLink() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * Returns the time of the last modification in seconds since epoch
     * (01.01.1970).
     *
     * return
     *      the last modification time.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the NSEntry is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public long getMTime() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * Copies this entry to another part of the namespace.
     *
     * param target
     *      the name to copy to.
     * param flags
     *      defining the operation modus.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL contains an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSEntry is already closed or the DEREFERENCE
     *      flag is given and dereferencing is impossible.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified target URL already exists, and the
     *      <code>OVERWRITE</code> flag is not given.
     * exception DoesNotExistException
     *      is thrown if the target lies in a non-existing part of the
     *      name space, unless the CREATEPARENTS flag is given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void copy(URL target, int flags) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException,
            IncorrectURLException;

    /**
     * Copies this entry to another part of the namespace.
     *
     * param target
     *      the name to copy to.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL contains an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSEntry is already closed or the DEREFERENCE
     *      flag is given and dereferencing is impossible.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified target URL already exists, and the
     *      <code>OVERWRITE</code> flag is not given.
     * exception DoesNotExistException
     *      is thrown if the target lies in a non-existing part of the
     *      name space, unless the CREATEPARENTS flag is given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void copy(URL target) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException,
            IncorrectURLException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL contains an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSEntry is already closed or the DEREFERENCE
     *      flag is given and dereferencing is impossible.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified target URL already exists, and the
     *      <code>OVERWRITE</code> flag is not given.
     * exception DoesNotExistException
     *      is thrown if the target lies in a non-existing part of the
     *      name space, unless the CREATEPARENTS flag is given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void link(URL target, int flags) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException, DoesNotExistException,
            IncorrectStateException, AlreadyExistsException, TimeoutException,
            NoSuccessException, IncorrectURLException;

    /**
     * Creates a symbolic link from the target to this entry.
     *
     * param target
     *      the name that will have the symbolic link to this entry.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL contains an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSEntry is already closed or the DEREFERENCE
     *      flag is given and dereferencing is impossible.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified target URL already exists, and the
     *      <code>OVERWRITE</code> flag is not given.
     * exception DoesNotExistException
     *      is thrown if the target lies in a non-existing part of the
     *      name space, unless the CREATEPARENTS flag is given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void link(URL target) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException, DoesNotExistException,
            IncorrectStateException, AlreadyExistsException, TimeoutException,
            NoSuccessException, IncorrectURLException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL contains an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSEntry is already closed or the DEREFERENCE
     *      flag is given and dereferencing is impossible.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified target URL already exists, and the
     *      <code>OVERWRITE</code> flag is not given.
     * exception DoesNotExistException
     *      is thrown if the target lies in a non-existing part of the
     *      name space, unless the CREATEPARENTS flag is given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void move(URL target, int flags) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException,
            IncorrectURLException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL contains an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSEntry is already closed or the DEREFERENCE
     *      flag is given and dereferencing is impossible.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified target URL already exists, and the
     *      <code>OVERWRITE</code> flag is not given.
     * exception DoesNotExistException
     *      is thrown if the target lies in a non-existing part of the
     *      name space, unless the CREATEPARENTS flag is given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void move(URL target) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException,
            IncorrectURLException;

    /**
     * Removes this entry and closes it.
     *
     * param flags
     *      defining the operation modus. The only allowed flags are
     *      RECURSIVE, DEREFERENCE, and NONE.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown if the entry is a directory and the RECURSIVE flag is not set,
     *      or the entry is a directory and the RECURSIVE flag is set,
     *      or invalid flags are specified.
     * exception IncorrectStateException
     *      is thrown when the NSEntry is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void remove(int flags) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException;

    /**
     * Removes this entry and closes it.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown if the entry is a directory and the RECURSIVE flag is not set,
     *      or the entry is a directory and the RECURSIVE flag is set,
     *      or invalid flags are set.
     * exception IncorrectStateException
     *      is thrown when the NSEntry is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void remove() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException;

    /**
     * Closes this entry. This is a non-blocking close. Any subsequent method
     * invocation on the object (except for close()) will throw an IncorrectState exception.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void close() throws NotImplementedException,
            NoSuccessException;

    /**
     * Closes this entry. Any subsequent method invocation on the object will
     * throw an IncorrectState exception.
     *
     * param timeoutInSeconds
     *      seconds to wait.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void close(float timeoutInSeconds) throws NotImplementedException,
            NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the given id is unknown or not supported, or illegal
     *      flags are specified, or RECURSIVE is specified on a non-directory.
     * exception IncorrectStateException
     *      is thrown when the NSEntry is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void permissionsAllow(String id, int permissions, int flags)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            IncorrectStateException, BadParameterException, TimeoutException,
            NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the given id is unknown or not supported, or illegal
     *      flags are specified, or RECURSIVE is specified on a non-directory.
     * exception IncorrectStateException
     *      is thrown when the NSEntry is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void permissionsDeny(String id, int permissions, int flags)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, IncorrectStateException,
            PermissionDeniedException, BadParameterException, TimeoutException,
            NoSuccessException;

    //
    // Task versions ...
    //

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

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

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

    /**
     * Creates a task that returns the last modification time of this entry.
     *
     * param mode
     *            the task mode.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<NSEntry, Long> getMTime(TaskMode mode)
            throws NotImplementedException;

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

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<NSEntry, Boolean> isEntry(TaskMode mode)
            throws NotImplementedException;

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

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<NSEntry, URL> readLink(TaskMode mode)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<NSEntry, Void> copy(TaskMode mode, URL target, int flags)
```

```
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSEntry, Void> copy(TaskMode mode, URL target)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSEntry, Void> link(TaskMode mode, URL target, int flags)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSEntry, Void> link(TaskMode mode, URL target)
            throws NotImplementedException;
    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<NSEntry, Void> move(TaskMode mode, URL target, int flags)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<NSEntry, Void> move(TaskMode mode, URL target)
            throws NotImplementedException;

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

    public Task<NSEntry, Void> remove(TaskMode mode)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSEntry, Void> close(TaskMode mode)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSEntry, Void> close(TaskMode mode, float timeoutInSeconds)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSEntry, Void> permissionsAllow(TaskMode mode, String id,
            int permissions, int flags) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<NSEntry, Void> permissionsDeny(TaskMode mode, String id,
            int permissions, int flags) throws NotImplementedException;
}
```

4.2.3 NSFactory

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

import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.AlreadyExistsException;
import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.DoesNotExistException;
import org.ogf.saga.error.IncorrectURLException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;
import org.ogf.saga.session.Session;
import org.ogf.saga.session.SessionFactory;
import org.ogf.saga.task.TaskMode;
import org.ogf.saga.task.Task;
import org.ogf.saga.url.URL;

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

    private static NSFactory getFactory(String sagaFactoryName)
    throws NoSuccessException, NotImplementedException {
return ImplementationBootstrapLoader.getNamespaceFactory(sagaFactoryName);
    }

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    protected abstract Task<NSFactory, NSEntry> doCreateNSEntry(TaskMode mode,
            Session session, URL name, int flags)
            throws NotImplementedException;

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

    /**
     * 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 NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, AlreadyExistsException, TimeoutException,
            NoSuccessException;

    /**
     * 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 NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, DoesNotExistException,
            AlreadyExistsException, TimeoutException, NoSuccessException;

    /**
     * Creates a namespace entry.
     *
     * param session
     *      the session handle.
     * param name
     *      the initial working directory.
     * param flags
     *      the open mode.
     * return
     *      the namespace entry.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static NSEntry createNSEntry(Session session, URL name, int flags)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, AlreadyExistsException, TimeoutException,
            NoSuccessException {
        return createNSEntry((String) null, session, name, flags);
    }

    /**
     * Creates a namespace entry.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param session
     *      the session handle.
     * param name
     *      the initial working directory.
     * param flags
     *      the open mode.
     * return
     *      the namespace entry.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static NSEntry createNSEntry(String sagaFactoryClassname, Session session, URL name, int fl
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, AlreadyExistsException, TimeoutException,
            NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).doCreateNSEntry(session, name, flags);
    }


    /**
     * Creates a namespace entry.
     *
     * param session
     *      the session handle.
     * param name
     *      the initial working directory.
     * return
     *      the namespace entry.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static NSEntry createNSEntry(Session session, URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, AlreadyExistsException, TimeoutException,
            NoSuccessException {
        return createNSEntry(session, name, Flags.READ.getValue());
    }

    /**
     * Creates a namespace entry.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param session
     *      the session handle.
     * param name
     *      the initial working directory.
     * return
     *      the namespace entry.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static NSEntry createNSEntry(String sagaFactoryClassname, Session session, URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, AlreadyExistsException, TimeoutException,
            NoSuccessException {
        return createNSEntry(sagaFactoryClassname, session, name, Flags.READ.getValue());
    }

    /**
     * Creates a namespace entry using the default session.
     *
     * param name
     *      the initial working directory.
     * param flags
     *      the open mode.
     * return
     *      the namespace entry.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static NSEntry createNSEntry(URL name, int flags)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, AlreadyExistsException, TimeoutException,
            NoSuccessException {
        return createNSEntry((Session) null, name, flags);
    }

    /**
     * Creates a namespace entry using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param name
     *      the initial working directory.
     * param flags
     *      the open mode.
     * return
     *      the namespace entry.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static NSEntry createNSEntry(String sagaFactoryClassname, URL name, int flags)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, AlreadyExistsException, TimeoutException,
            NoSuccessException {
        return createNSEntry(sagaFactoryClassname, (Session) null, name, flags);
    }

    /**
     * Creates a namespace entry using the default session.
     *
     * param name
     *      the initial working directory.
     * return
     *      the namespace entry.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static NSEntry createNSEntry(URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, AlreadyExistsException, TimeoutException,
            NoSuccessException {
        return createNSEntry((Session) null, name);
    }


    /**
     * Creates a namespace entry using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param name
     *      the initial working directory.
     * return
     *      the namespace entry.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static NSEntry createNSEntry(String sagaFactoryClassname, URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, AlreadyExistsException, TimeoutException,
            NoSuccessException {
        return createNSEntry(sagaFactoryClassname, (Session) null, name);
    }

    /**
     * Creates a namespace directory.
     *
     * param session
     *      the session handle.
     * param name
     *      the initial working directory.
     * param flags
     *      the open mode.
     * return
     *      the namespace directory.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static NSDirectory createNSDirectory(Session session, URL name,
            int flags) throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, AlreadyExistsException, TimeoutException,
            NoSuccessException {
        return createNSDirectory((String) null, session, name, flags);
    }

    /**
```

```
     * Creates a namespace directory.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param session
     *      the session handle.
     * param name
     *      the initial working directory.
     * param flags
     *      the open mode.
     * return
     *      the namespace directory.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static NSDirectory createNSDirectory(String sagaFactoryClassname, Session session, URL name
            int flags) throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, AlreadyExistsException, TimeoutException,
            NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).doCreateNSDirectory(session, name, flags);
    }


    /**
     * Creates a namespace directory.
     *
     * param session
     *      the session handle.
     * param name
     *      the initial working directory.
     * return
     *      the namespace directory.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static NSDirectory createNSDirectory(Session session, URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, AlreadyExistsException, TimeoutException,
            NoSuccessException {
        return createNSDirectory(session, name, Flags.READ.getValue());
    }

    /**
     * Creates a namespace directory.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param session
     *      the session handle.
     * param name
     *      the initial working directory.
     * return
     *      the namespace directory.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static NSDirectory createNSDirectory(String sagaFactoryClassname, Session session, URL name
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, AlreadyExistsException, TimeoutException,
            NoSuccessException {
        return createNSDirectory(sagaFactoryClassname, session, name, Flags.READ.getValue());
    }

    /**
     * Creates a namespace directory using the default session.
     *
     * param name
     *      the initial working directory.
     * param flags
     *      the open mode.
     * return
     *      the namespace directory.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static NSDirectory createNSDirectory(URL name, int flags)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, AlreadyExistsException, TimeoutException,
            NoSuccessException {
        return createNSDirectory((Session) null, name, flags);
    }

    /**
     * Creates a namespace directory using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param name
     *      the initial working directory.
     * param flags
     *      the open mode.
     * return
     *      the namespace directory.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static NSDirectory createNSDirectory(String sagaFactoryClassname, URL name, int flags)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, AlreadyExistsException, TimeoutException,
            NoSuccessException {
        return createNSDirectory(sagaFactoryClassname, (Session) null, name, flags);
    }

    /**
     * Creates a namespace directory using the default session.
     *
     * param name
     *      the initial working directory.
     * return
     *      the namespace directory.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static NSDirectory createNSDirectory(URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, AlreadyExistsException, TimeoutException,
            NoSuccessException {
        return createNSDirectory(name, Flags.READ.getValue());
    }

    /**
     * Creates a namespace directory using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param name
     *      the initial working directory.
     * return
     *      the namespace directory.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static NSDirectory createNSDirectory(String sagaFactoryClassname,  URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, AlreadyExistsException, TimeoutException,
            NoSuccessException {
        return createNSDirectory(sagaFactoryClassname, name, Flags.READ.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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<NSFactory, NSEntry> createNSEntry(TaskMode mode,
            Session session, URL name, int flags)
            throws NotImplementedException, NoSuccessException {
        return createNSEntry(null, mode, session, name, flags);
    }

    /**
     * Creates a task that creates a namespace entry.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<NSFactory, NSEntry> createNSEntry(String sagaFactoryClassname, TaskMode mode,
            Session session, URL name, int flags)
            throws NotImplementedException, NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     * throws NoSuccessException
     *            is thrown when the Saga factory could not be created.
     */
    public static Task<NSFactory, NSEntry> createNSEntry(TaskMode mode,
            Session session, URL name) throws NotImplementedException,
            NoSuccessException {
        return createNSEntry(mode, session, name, Flags.READ.getValue());
    }


    /**
     * Creates a task that creates a namespace entry.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *           the task mode.
     * param session
     *           the session handle.
     * param name
     *           the initial working directory.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     * throws NoSuccessException
     *            is thrown when the Saga factory could not be created.
     */
    public static Task<NSFactory, NSEntry> createNSEntry(String sagaFactoryClassname, TaskMode mode,
            Session session, URL name) throws NotImplementedException,
            NoSuccessException {
        return createNSEntry(sagaFactoryClassname, mode, session, name, Flags.READ.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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *                when the Saga factory could not be created.
     */
    public static Task<NSFactory, NSEntry> createNSEntry(TaskMode mode,
            URL name, int flags) throws NotImplementedException,
            NoSuccessException {
        return createNSEntry(mode, null, name, flags);
    }

    /**
     * Creates a task that creates a namespace entry using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param name
     *            the initial working directory.
     * param flags
     *            the open mode.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *                when the Saga factory could not be created.
     */
    public static Task<NSFactory, NSEntry> createNSEntry(String sagaFactoryClassname, TaskMode mode,
            URL name, int flags) throws NotImplementedException,
            NoSuccessException {
        return createNSEntry(sagaFactoryClassname, mode, null, 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *               when the Saga factory could not be created.
     */
    public static Task<NSFactory, NSEntry> createNSEntry(TaskMode mode, URL name)
            throws NotImplementedException, NoSuccessException {
        return createNSEntry(mode, null, name);
    }

    /**
     * Creates a task that creates a namespace entry using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param name
     *            the initial working directory.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *                when the Saga factory could not be created.
     */
    public static Task<NSFactory, NSEntry> createNSEntry(String sagaFactoryClassname, TaskMode mode, U
            throws NotImplementedException, NoSuccessException {
        return createNSEntry(sagaFactoryClassname, mode, null, name);
    }

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */

    public static Task<NSFactory, NSDirectory> createNSDirectory(TaskMode mode,
            Session session, URL name, int flags)
            throws NotImplementedException, NoSuccessException {
        return createNSDirectory(null, mode, session, name, flags);
    }

    /**
     * Creates a task that creates a namespace directory.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */

    public static Task<NSFactory, NSDirectory> createNSDirectory(String sagaFactoryClassname, TaskMode
            Session session, URL name, int flags)
            throws NotImplementedException, NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
```

```
public static Task<NSFactory, NSDirectory> createNSDirectory(TaskMode mode,
        Session session, URL name) throws NotImplementedException,
        NoSuccessException {
    return createNSDirectory(mode, session, name, Flags.READ.getValue());
}

/**
 * Creates a task that creates a namespace directory.
 *
 * param sagaFactoryClassname
 *      the class name of the Saga factory to be used.
 * param mode
 *           the task mode.
 * param session
 *           the session handle.
 * param name
 *           the initial working directory.
 * return the task.
 * exception NotImplementedException
 *               is thrown when the task version of this method is not
 *               implemented.
 * throws NoSuccessException
 *            is thrown when the Saga factory could not be created.
 */
public static Task<NSFactory, NSDirectory> createNSDirectory(String sagaFactoryClassname, TaskMode
        Session session, URL name) throws NotImplementedException,
        NoSuccessException {
    return createNSDirectory(sagaFactoryClassname, mode, session, name, Flags.READ.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 NotImplementedException
 *               is thrown when the task version of this method is not
 *               implemented.
 * exception NoSuccessException
 *               is thrown when the default session could not be created or
 *               when the Saga factory could not be created.
 */
public static Task<NSFactory, NSDirectory> createNSDirectory(TaskMode mode,
        URL name, int flags) throws NotImplementedException,
        NoSuccessException {
    return createNSDirectory(mode, null, name, flags);
}

/**
 * Creates a task that creates a namespace directory using the default
 * session.
 *
 * param sagaFactoryClassname
 *      the class name of the Saga factory to be used.
 * param mode
 *           the task mode.
 * param name
 *           the initial working directory.
 * param flags
 *           the open mode.
 * return the task.
 * exception NotImplementedException
 *               is thrown when the task version of this method is not
 *               implemented.
 * exception NoSuccessException
 *               is thrown when the default session could not be created or
 *               when the Saga factory could not be created.
 */
public static Task<NSFactory, NSDirectory> createNSDirectory(String sagaFactoryClassname, TaskMode
        URL name, int flags) throws NotImplementedException,
        NoSuccessException {
    return createNSDirectory(sagaFactoryClassname, mode, null, 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 NotImplementedException
 *               is thrown when the task version of this method is not
 *               implemented.
 * exception NoSuccessException
 *               is thrown when the default session could not be created or
 *               when the Saga factory could not be created.
 */
public static Task<NSFactory, NSDirectory> createNSDirectory(TaskMode mode,
        URL name) throws NotImplementedException, NoSuccessException {
    return createNSDirectory(mode, null, name);
}
/**
 * Creates a task that creates a namespace directory using the default
 * session.
 *
 * param sagaFactoryClassname
 *      the class name of the Saga factory to be used.
 * param mode
 *            the task mode.
 * param name
 *            the initial working directory.
 * return the task.
 * exception NotImplementedException
 *                is thrown when the task version of this method is not
 *                implemented.
 * exception NoSuccessException
 *                is thrown when the default session could not be created or
 *                when the Saga factory could not be created.
 */
public static Task<NSFactory, NSDirectory> createNSDirectory(String sagaFactoryClassname, TaskMode
        URL name) throws NotImplementedException, NoSuccessException {
    return createNSDirectory(sagaFactoryClassname, mode, null, name);
}
}
```

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 {
    /** Indicates the absence of flags. */
    NONE(0),
    /**
```

```
     * Enforces an operation which creates a new namespace entry to continue
     * even if the target entry does not already exist.
     */
    OVERWRITE(1),
    /** Enforces an operation to apply recursively on a directory tree. */
    RECURSIVE(2),
    /**
     * Enforces an operation to apply not to the entry pointed to by the target name,
     * but to the link target of that entry.
     */
    DEREFERENCE(4),
    /**
     * Allows a namespace entry to be created while opening it, if it does not
     * already exist.
     */
    CREATE(8),
    /**
     * If the entry already exists, the {link #CREATE} flag is not silenty
     * ignored. Instead, an {link org.ogf.saga.error.AlreadyExistsException AlreadyExistsException}
     * is raised.
     */
    EXCL(16),
    /** Enforces a lock on the namespace entry when it is opened. */
    LOCK(32),
    /** Implies that missing path elements are created on the fly. Implies CREATE. */
    CREATEPARENTS(64),
    /** Upon opening, the file is truncated to length 0. Implies WRITE. */
    TRUNCATE(128),
    /** Upon opening, the file pointer is set to the end of the file. IMPLIES WRITE. */
    APPEND(256),
    /** The file or directory is opened for reading. */
    READ(512),
    /** The file or directory is opened for writing. */
    WRITE(1024),
    /** The file or directory is opened for reading and writing. */
    READWRITE(512 | 1024),
    /** All flags applicable to the namespace package. */
    ALLNAMESPACEFLAGS(1 | 2 | 4 | 8 | 16 | 32 | 64 | 512 | 1024),
    /** For OS-es that distinguish between binary and non-binary modes. */
    BINARY(2048),
    /** All flags applicable to the logical file package. */
    ALLLOGICALFILEFLAGS(ALLNAMESPACEFLAGS.getValue()),
    /** All flags applicable to the file package. */
    ALLFILEFLAGS(ALLNAMESPACEFLAGS.getValue() | 128 | 256 | 2048);

    private int value;

    private 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.

A `File` represents an open file descriptor for read/write operations on a physical file. A deviation from the language-independent SAGA specifications is that the read and write methods do not return POSIX error codes in case of an error. Instead, they throw a `SagaIOException`. If a POSIX error code happens to be available, it is stored inside the exception object.

A `Directory` extends an `NSDirectory` with methods to open files and other directories. Some methods have been renamed slightly to avoid conflicts with methods in `NSDirectory`, as only the type of the return value differs, and Java does not allow that.

4.3.1 Directory

```
package org.ogf.saga.file;

import org.ogf.saga.error.AlreadyExistsException;
import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.DoesNotExistException;
import org.ogf.saga.error.IncorrectStateException;
import org.ogf.saga.error.IncorrectURLException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;
import org.ogf.saga.namespace.NSDirectory;
import org.ogf.saga.task.TaskMode;
import org.ogf.saga.task.Task;
import org.ogf.saga.url.URL;

/**
 * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL contains an invalid entry name,
     *      or illegal flags are specified (the only legal flags are NONE
     *      and DEREFERENCE).
     * exception IncorrectStateException
     *      is thrown when the Directory is already closed.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception DoesNotExistException
     *      is thrown if the specified name does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public long getSize(URL name, int flags) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Returns the number of bytes in the specified file.
     *
     * param name
     *      name of file to inspect.
     * return
     *      the size.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL contains an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the Directory is already closed.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception DoesNotExistException
     *      is thrown if the specified name does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public long getSize(URL name) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL contains an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the Directory is already closed.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception DoesNotExistException
     *      is thrown if the specified name does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public boolean isFile(URL name) throws NotImplementedException,
            IncorrectURLException, DoesNotExistException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException;

    // 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL does not point to a directory,
     *      or is an invalid directory name.
     * exception IncorrectStateException
     *      is thrown when the Directory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public Directory openDirectory(URL name, int flags)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Creates a new <code>Directory</code> instance.
     *
     * param name
     *            directory to open.
     * return the opened directory instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL does not point to a directory,
     *      or is an invalid directory name.
     * exception IncorrectStateException
     *      is thrown when the Directory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      not thrown, but specified because a method may be invoked
     *      that can throw this exception, but will not in this case.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public Directory openDirectory(URL name) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * Creates a new <code>File</code> instance.
     *
     * param name
     *      file to open.
     * param flags
     *      defining the operation modus.
     * return
     *      the opened file instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public File openFile(URL name, int flags) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * Creates a new <code>File</code> instance.
     *
     * param name
     *      file to open.
     * return
     *      the opened file instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public File openFile(URL name) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * Creates a new <code>FileInputStream</code> instance.
     *
     * param name
     *      file to open.
     * return
     *      the input stream.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public FileInputStream openFileInputStream(URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Creates a new <code>FileOutputStream</code> instance.
     *
     * param name
     *      file to open.
     * return
     *      the output stream.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL points to a directory,
     *      or is an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the parent directory does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public FileOutputStream openFileOutputStream(URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL points to a directory,
     *      or is an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the NSDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the parent directory does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public FileOutputStream openFileOutputStream(URL name, boolean append)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    //
    // 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<Directory, Long> getSize(TaskMode mode, URL name, int flags)
            throws NotImplementedException;
    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<Directory, Long> getSize(TaskMode mode, URL name)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<Directory, Boolean> isFile(TaskMode mode, URL name)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<Directory, Directory> openDirectory(TaskMode mode, URL name,
            int flags) throws NotImplementedException;

    /**
```

```
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<Directory, Directory> openDirectory(TaskMode mode, URL name)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<Directory, File> openFile(TaskMode mode, URL name, int flags)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<Directory, File> openFile(TaskMode mode, URL name)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<Directory, FileInputStream> openFileInputStream(TaskMode mode,
            URL name) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<Directory, FileOutputStream> openFileOutputStream(
            TaskMode mode, URL name) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<Directory, FileOutputStream> openFileOutputStream(
            TaskMode mode, URL name, boolean append)
            throws NotImplementedException;

}
```

4.3.2 File

```
package org.ogf.saga.file;

import java.util.List;

import org.ogf.saga.buffer.Buffer;
import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.IncorrectStateException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.SagaIOException;
import org.ogf.saga.error.TimeoutException;
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 SagaIOException, not a POSIX error code.
 */
public interface File extends NSEntry {

    // Inspection

    /**
     * Returns the number of bytes in the file.
     *
     * return the size.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the Directory is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public long getSize() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    // 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
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or if the file was opened WriteOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown if the provided buffer is not large enough to hold
     *      the required length.
     * exception IncorrectStateException
     *      is thrown when the File is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception SagaIOException
     *      is thrown on read failures when the SAGA specifications say that
     *      a POSIX error number should be returned (which does not really
     *      make sense for Java).
     */
    public int read(Buffer buffer, int len) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException,
            SagaIOException;

    /**
     * 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
     *      the offset in the buffer.
     * param len
     *      the number of bytes to be read.
     * return
     *      the number of bytes read.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or if the file was opened WriteOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown if the provided buffer is not large enough to hold
     *      the required length.
     * exception IncorrectStateException
     *      is thrown when the File is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception SagaIOException
     *      is thrown on read failures when the SAGA specifications say that
     *      a POSIX error number should be returned (which does not really
     *      make sense for Java).
     */
    public int read(Buffer buffer, int offset, int len)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException, TimeoutException,
            NoSuccessException, SagaIOException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or if the file was opened WriteOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown if the provided buffer is not large enough to hold
     *      the required length.
     * exception IncorrectStateException
     *      is thrown when the File is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception SagaIOException
     *      is thrown on read failures when the SAGA specifications say that
     *      a POSIX error number should be returned (which does not really
     *      make sense for Java).
     */
    public int read(Buffer buffer) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException,
            SagaIOException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or if the file was opened ReadOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown if the provided buffer is not large enough to hold
     *      the specified length.
     * exception IncorrectStateException
     *      is thrown when the File is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception SagaIOException
     *      is thrown on write failures when the SAGA specifications say that
     *      a POSIX error number should be returned (which does not really
     *      make sense for Java).
     */
    public int write(Buffer buffer, int offset, int len)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException, TimeoutException,
            NoSuccessException, SagaIOException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or if the file was opened ReadOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown if the provided buffer is not large enough to hold
     *      the specified length.
     * exception IncorrectStateException
     *      is thrown when the File is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception SagaIOException
     *      is thrown on write failures when the SAGA specifications say that
     *      a POSIX error number should be returned (which does not really
     *      make sense for Java).
     */
    public int write(Buffer buffer, int len) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException,
            SagaIOException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or if the file was opened ReadOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown if the provided buffer is not large enough to hold
     *      the specified length.
     * exception IncorrectStateException
     *      is thrown when the File is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception SagaIOException
     *      is thrown on write failures when the SAGA specifications say that
     *      a POSIX error number should be returned (which does not really
     *      make sense for Java).
     */
    public int write(Buffer buffer) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException,
            SagaIOException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the File is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception SagaIOException
     *      is thrown on seek failures when the SAGA specifications say that
     *      a POSIX error number should be returned (which does not really
     *      make sense for Java).
     */
    public long seek(long offset, SeekMode whence)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            IncorrectStateException, TimeoutException, NoSuccessException,
            SagaIOException;

    // Scattered I/O

    /**
     * Gather/scatter read.
     *
     * param iovecs
     *      array of IOVecs determining how much to read and where to
     *      store it.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or if the file was opened WriteOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown if the provided buffer is not large enough to hold
     *      the required length.
     * exception IncorrectStateException
     *      is thrown when the File is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception SagaIOException
     *      is thrown on read failures when the SAGA specifications say that
     *      a POSIX error number should be returned (which does not really
     *      make sense for Java).
     */
    public void readV(IOVec[] iovecs) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException,
            SagaIOException;

    /**
     * Gather/scatter write.
     *
     * param iovecs
     *      array of IOVecs determining how much to write and where to
     *      obtain the data from.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or if the file was opened ReadOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown if the provided buffer is not large enough to hold
     *      the specified length.
     * exception IncorrectStateException
     *      is thrown when the File is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception SagaIOException
     *      is thrown on write failures when the SAGA specifications say that
     *      a POSIX error number should be returned (which does not really
     *      make sense for Java).
     */
    public void writeV(IOVec[] iovecs) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException,
            SagaIOException;

    // Pattern-based I/O

    /**
     * Determines the storage size required for a pattern I/O operation.
     *
     * param pattern
     *      to determine size for.
     * return
     *      the size.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the pattern cannot be parsed.
     * exception IncorrectStateException
     *      is thrown when the File is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public int sizeP(String pattern) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            IncorrectStateException, PermissionDeniedException,
            BadParameterException, TimeoutException, NoSuccessException;

    /**
     * Pattern-based read.
     *
     * param pattern
     *      specification for the read operation.
     * param buffer
     *      to store data into.
     * return
     *      number of succesfully read bytes.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or if the file was opened WriteOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the pattern cannot be parsed.
     * exception IncorrectStateException
     *      is thrown when the File is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception SagaIOException
     *      is thrown on read failures when the SAGA specifications say that
     *      a POSIX error number should be returned (which does not really
     *      make sense for Java).
     */
    public int readP(String pattern, Buffer buffer)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException, TimeoutException,
            NoSuccessException, SagaIOException;

    /**
     * Pattern-based write.
     *
     * param pattern
     *      specification for the write operation.
     * param buffer
     *      to be written.
     * return
     *      number of succesfully written bytes.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or if the file was opened ReadOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the pattern cannot be parsed.
     * exception IncorrectStateException
     *      is thrown when the File is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception SagaIOException
     *      is thrown on write failures when the SAGA specifications say that
     *      a POSIX error number should be returned (which does not really
     *      make sense for Java).
     */
    public int writeP(String pattern, Buffer buffer)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException, TimeoutException,
            NoSuccessException, SagaIOException;

    // extended I/O

    /**
     * Lists the extended modes available in this implementation and/or on the
     * server side.
     *
     * return
     *      list of available modes.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the File is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public List<String> modesE() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the emode is not supported or the spec cannot
     *      be parsed.
     * exception IncorrectStateException
     *      is thrown when the File is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public int sizeE(String emode, String spec) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            IncorrectStateException, PermissionDeniedException,
            BadParameterException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or if the file was opened WriteOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the emode is not supported or the spec cannot
     *      be parsed.
     * exception IncorrectStateException
     *      is thrown when the File is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception SagaIOException
     *      is thrown on read failures when the SAGA specifications say that
     *      a POSIX error number should be returned (which does not really
     *      make sense for Java).
     */
    public int readE(String emode, String spec, Buffer buffer)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException, TimeoutException,
            NoSuccessException, SagaIOException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or if the file was opened ReadOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the emode is not supported or the spec cannot
     *      be parsed.
     * exception IncorrectStateException
     *      is thrown when the File is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception SagaIOException
     *      is thrown on write failures when the SAGA specifications say that
     *      a POSIX error number should be returned (which does not really
     *      make sense for Java).
     */
    public int writeE(String emode, String spec, Buffer buffer)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException, TimeoutException,
            NoSuccessException, SagaIOException;

    //
    // 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<File, Long> getSize(TaskMode mode)
            throws NotImplementedException;

    // 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */

    public Task<File, Integer> read(TaskMode mode, Buffer buffer, int offset,
            int len) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<File, Integer> read(TaskMode mode, Buffer buffer, int len)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<File, Integer> read(TaskMode mode, Buffer buffer)
            throws NotImplementedException;
    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<File, Integer> write(TaskMode mode, Buffer buffer, int offset,
            int len) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<File, Integer> write(TaskMode mode, Buffer buffer, int len)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<File, Integer> write(TaskMode mode, Buffer buffer)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<File, Long> seek(TaskMode mode, long offset, SeekMode whence)
            throws NotImplementedException;

    // 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<File, Void> readV(TaskMode mode, IOVec[] iovecs)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<File, Void> writeV(TaskMode mode, IOVec[] iovecs)
            throws NotImplementedException;

    // 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<File, Integer> sizeP(TaskMode mode, String pattern)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<File, Integer> readP(TaskMode mode, String pattern,
            Buffer buffer) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<File, Integer> writeP(TaskMode mode, String pattern,
            Buffer buffer) throws NotImplementedException;

    // 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<File, List<String>> modesE(TaskMode mode)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<File, Integer> sizeE(TaskMode mode, String emode, String spec)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<File, Integer> readE(TaskMode mode, String emode, String spec,
            Buffer buffer) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<File, Integer> writeE(TaskMode mode, String emode, String spec,
            Buffer buffer) throws NotImplementedException;
}
```

4.3.3 IOVec

```
package org.ogf.saga.file;

import org.ogf.saga.buffer.Buffer;
import org.ogf.saga.error.BadParameterException;

/**
 * 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.
     * exception BadParameterException
     *           is thrown when the lenIn is set to an illegal
     *           value (< 0 or larger than size if size != -1).
     */
    void setLenIn(int len) throws BadParameterException;

    /**
     * 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.
     * exception BadParameterException
     *           is thrown when the offset is set to an illegal
     *           value (< 0).
     */
    void setOffset(int offset) throws BadParameterException;

    /**
     * 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.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.AlreadyExistsException;
import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.DoesNotExistException;
import org.ogf.saga.error.IncorrectURLException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;
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;
import org.ogf.saga.url.URL;

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

    private static FileFactory getFactory(String sagaFactoryName)
            throws NoSuccessException {
        return ImplementationBootstrapLoader.getFileFactory(sagaFactoryName);
    }

    /**
     * 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 BadParameterException, NoSuccessException;

    /**
     * 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 BadParameterException, NoSuccessException;

    /**
     * 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 NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * 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 NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * 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 NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * 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 NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    protected abstract Task<FileFactory, File> doCreateFile(TaskMode mode,
            Session session, URL name, int flags)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    protected abstract Task<FileFactory, FileInputStream> doCreateFileInputStream(
            TaskMode mode, Session session, URL name)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    protected abstract Task<FileFactory, FileOutputStream> doCreateFileOutputStream(
            TaskMode mode, Session session, URL name, boolean append)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    protected abstract Task<FileFactory, Directory> doCreateDirectory(
            TaskMode mode, Session session, URL name, int flags)
            throws NotImplementedException;

    /**
     * Creates an IOVec.
     *
     * param data
     *      data to be used.
     * param lenIn
     *      number of bytes to read/write on readV/writeV.
     * return
     *      the IOVec.
     * exception BadParameterException
     *      is thrown when <code>lenIn</code> is larger than the size of the
     *      specified buffer, or < 0, or when the implementation cannot handle
     *      the specified data buffer.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static IOVec createIOVec(byte[] data, int lenIn)
            throws BadParameterException, NoSuccessException {
return createIOVec(null, data, lenIn);
    }

    /**
     * Creates an IOVec.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param data
     *      data to be used.
     * param lenIn
     *      number of bytes to read/write on readV/writeV.
     * return
     *      the IOVec.
     * exception BadParameterException
     *      is thrown when <code>lenIn</code> is larger than the size of the
     *      specified buffer, or < 0, or when the implementation cannot handle
     *      the specified data buffer.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static IOVec createIOVec(String sagaFactoryClassname, byte[] data, int lenIn)
            throws BadParameterException, NoSuccessException {
return getFactory(sagaFactoryClassname).doCreateIOVec(data, lenIn);
    }
    /**
     * Creates an IOVec.
     *
     * param data
     *      data to be used.
     * return
     *      the IOVec.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception BadParameterException
     *      is thrown when the implementation cannot handle
     *      the specified data buffer.
     */
    public static IOVec createIOVec(byte[] data) throws BadParameterException,
            NoSuccessException {
        return createIOVec(data, data.length);
    }


    /**
     * Creates an IOVec.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param data
     *      data to be used.
     * return
     *      the IOVec.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception BadParameterException
     *      is thrown when the implementation cannot handle
     *      the specified data buffer.
     */
    public static IOVec createIOVec(String sagaFactoryClassname, byte[] data) throws BadParameterExcep
            NoSuccessException {
        return createIOVec(sagaFactoryClassname, data, data.length);
    }

    /**
     * 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.
     * exception BadParameterException
     *      is thrown when <code>lenIn</code> is larger than the size of the
     *      specified buffer, or < 0.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static IOVec createIOVec(int size, int lenIn)
            throws BadParameterException, NoSuccessException {
return createIOVec(null, size, lenIn);
    }

    /**
     * Creates an IOVec.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param size
     *      size of data to be used.
     * param lenIn
     *      number of bytes to read/write on readV/writeV.
     * return
     *      the IOVec.
     * exception BadParameterException
     *      is thrown when <code>lenIn</code> is larger than the size of the
     *      specified buffer, or < 0.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static IOVec createIOVec(String sagaFactoryClassname, int size, int lenIn)
            throws BadParameterException, NoSuccessException {
return getFactory(sagaFactoryClassname).doCreateIOVec(size, lenIn);
    }

    /**
     * Creates an IOVec.
     *
     * param size
     *      size of data to be used.
     * return
     *      the IOVec.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception BadParameterException
     *      is thrown when the implementation cannot handle the specified size.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static IOVec createIOVec(int size) throws BadParameterException,
            NoSuccessException, NotImplementedException {
return createIOVec(size, size);
    }

    /**
     * Creates an IOVec.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param size
     *      size of data to be used.
     * return
     *      the IOVec.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception BadParameterException
     *      is thrown when the implementation cannot handle the specified size.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static IOVec createIOVec(String sagaFactoryClassname, int size) throws BadParameterExceptio
            NoSuccessException, NotImplementedException {
return createIOVec(sagaFactoryClassname, size, size);
    }

    /**
     * Creates a File.
     *
     * param session
     *      the session handle.
     * param name
     *      location of the file.
     * param flags
     *      the open mode.
     * return
     *      the file instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static File createFile(Session session, URL name, int flags)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
return createFile((String) null, session, name, flags);
    }


    /**
     * Creates a File.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param session
     *      the session handle.
     * param name
     *      location of the file.
     * param flags
     *      the open mode.
     * return
     *      the file instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static File createFile(String sagaFactoryClassname, Session session, URL name, int flags)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).doCreateFile(session, name, flags);
    }

    /**
     * Creates a File for reading.
     *
     * param session
     *      the session handle.
     * param name
     *      location of the file.
     * return
     *      the file instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static File createFile(Session session, URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
return createFile(session, name, Flags.READ.getValue());
    }
    /**
     * Creates a File for reading.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param session
     *      the session handle.
     * param name
     *      location of the file.
     * return
     *      the file instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static File createFile(String sagaFactoryClassname, Session session, URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
return createFile(sagaFactoryClassname, 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static File createFile(URL name, int flags)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createFile((Session) null, name, flags);
    }


    /**
     * Creates a File using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param name
     *      location of the file.
     * param flags
     *      the open mode.
     * return
     *      the file instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static File createFile(String sagaFactoryClassname, URL name, int flags)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createFile(sagaFactoryClassname, (Session) null, name, flags);
    }

    /**
     * Creates a File for reading, using the default session.
     *
     * param name
     *      location of the file.
     * return
     *      the file instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static File createFile(URL name) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException {
        return createFile((Session) null, name);
    }


    /**
     * Creates a File for reading, using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param name
     *      location of the file.
     * return
     *      the file instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static File createFile(String sagaFactoryClassname, URL name) throws NotImplementedExceptic
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException {
        return createFile(sagaFactoryClassname, SessionFactory.createSession(sagaFactoryClassname), na
    }

    /**
     * Creates a FileInputStream.
     *
     * param session
     *      the session handle.
     * param name
     *      location of the file.
     * return
     *      the FileInputStream instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static FileInputStream createFileInputStream(Session session,
            URL name) throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createFileInputStream((String) null, session, name);
    }

    /**
     * Creates a FileInputStream.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param session
     *      the session handle.
     * param name
     *      location of the file.
     * return
     *      the FileInputStream instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static FileInputStream createFileInputStream(String sagaFactoryClassname, Session session,
            URL name) throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).doCreateFileInputStream(session, name);
    }

    /**
     * Creates a FileInputStream using the default session.
     *
     * param name
     *      location of the file.
     * return
     *      the FileInputStream instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static FileInputStream createFileInputStream(URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createFileInputStream((Session) null, name);
    }


    /**
     * Creates a FileInputStream using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param name
     *      location of the file.
     * return
     *      the FileInputStream instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static FileInputStream createFileInputStream(String sagaFactoryClassname, URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createFileInputStream(sagaFactoryClassname, SessionFactory.createSession(sagaFactoryCla
    }

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL points to a directory,
     *      or is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the parent directory does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static FileOutputStream createFileOutputStream(Session session,
            URL name, boolean append) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException {
        return createFileOutputStream((String) null, session, name, append);
    }

    /**
     * Creates a FileOutputStream.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL points to a directory,
     *      or is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the parent directory does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static FileOutputStream createFileOutputStream(String sagaFactoryClassname, Session sessior
            URL name, boolean append) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).doCreateFileOutputStream(session, name, append);
    }


    /**
     * Creates a FileOutputStream.
     *
     * param session
     *      the session handle.
     * param name
     *      location of the file.
     * return
     *      the FileOutputStream instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL points to a directory,
     *      or is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the parent directory does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static FileOutputStream createFileOutputStream(Session session,
            URL name) throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createFileOutputStream(session, name, false);
    }


    /**
     * Creates a FileOutputStream.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param session
     *      the session handle.
     * param name
     *      location of the file.
     * return
     *      the FileOutputStream instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL points to a directory,
     *      or is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the parent directory does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static FileOutputStream createFileOutputStream(String sagaFactoryClassname, Session sessio
            URL name) throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createFileOutputStream(sagaFactoryClassname, session, name, false);
    }

    /**
     * Creates a FileOutputStream using the default session.
     *
     * param name
     *      location of the file.
     * param append
     *      when set, the file is opened for appending.
     * return
     *      the FileOutputStream instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL points to a directory,
     *      or is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the parent directory does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static FileOutputStream createFileOutputStream(URL name,
            boolean append) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException {
        return createFileOutputStream((Session) null, name, append);
    }


    /**
     * Creates a FileOutputStream using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param name
     *      location of the file.
     * param append
     *      when set, the file is opened for appending.
     * return
     *      the FileOutputStream instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
```

```
 *      is thrown when a remote operation did not complete successfully
 *      because the network communication or the remote service timed
 *      out.
 * exception BadParameterException
 *      is thrown when the specified URL points to a directory,
 *      or is an invalid entry name.
 * exception IncorrectURLException
 *      is thrown if an implementation cannot handle the specified
 *      protocol, or that access to the specified entity via the
 *      given protocol is impossible.
 * exception AlreadyExistsException
 *      is not thrown, but a method may be invoked that may throw it (but
 *      not in this case).
 * exception DoesNotExistException
 *      is thrown if the parent directory does not exist.
 * exception NoSuccessException
 *      is thrown when the operation was not successfully performed,
 *      and none of the other exceptions apply.
 */
public static FileOutputStream createFileOutputStream(String sagaFactoryClassname, URL name,
        boolean append) throws NotImplementedException,
        IncorrectURLException, AuthenticationFailedException,
        AuthorizationFailedException, PermissionDeniedException,
        BadParameterException, AlreadyExistsException,
        DoesNotExistException, TimeoutException, NoSuccessException {
    Session session = SessionFactory.createSession(sagaFactoryClassname);
    return createFileOutputStream(sagaFactoryClassname, session, name, append);
}


/**
 * Creates a FileOutputStream using the default session.
 *
 * param name
 *      location of the file.
 * return
 *      the FileOutputStream instance.
 * exception NotImplementedException
 *      is thrown if the implementation does not provide an
 *      implementation of this method.
 * exception PermissionDeniedException
 *      is thrown when the method failed because the identity used did
 *      not have sufficient permissions to perform the operation
 *      successfully.
 * exception AuthorizationFailedException
 *      is thrown when none of the available contexts of the
 *      used session could be used for successful authorization.
 *      This error indicates that the resource could not be accessed
 *      at all, and not that an operation was not available due to
 *      restricted permissions.
```

```
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL points to a directory,
     *      or is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the parent directory does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static FileOutputStream createFileOutputStream(URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createFileOutputStream(name, false);
    }

    /**
     * Creates a FileOutputStream using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param name
     *      location of the file.
     * return
     *      the FileOutputStream instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL points to a directory,
     *      or is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the parent directory does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static FileOutputStream createFileOutputStream(String sagaFactoryClassname, URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createFileOutputStream(sagaFactoryClassname, name, false);
    }


    /**
     * Creates a Directory.
     *
     * param session
     *      the session handle.
     * param name
     *      location of the directory.
     * param flags
     *      the open mode.
     * return
     *      the directory instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static Directory createDirectory(Session session, URL name, int flags)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createDirectory((String) null, session, name, flags);
    }


    /**
     * Creates a Directory.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param session
     *      the session handle.
     * param name
     *      location of the directory.
     * param flags
     *      the open mode.
     * return
     *      the directory instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static Directory createDirectory(String sagaFactoryClassname, Session session, URL name, in
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).doCreateDirectory(session, name, flags);
    }

    /**
     * Creates a Directory for reading.
     *
     * param session
     *      the session handle.
     * param name
     *      location of the directory.
     * return
     *      the directory instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static Directory createDirectory(Session session, URL name)
```

```
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createDirectory(session, name, Flags.READ.getValue());
    }


    /**
     * Creates a Directory for reading.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param session
     *      the session handle.
     * param name
     *      location of the directory.
     * return
     *      the directory instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static Directory createDirectory(String sagaFactoryClassname, Session session, URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createDirectory(sagaFactoryClassname, 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static Directory createDirectory(URL name, int flags)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createDirectory((Session) null, name, flags);
    }


    /**
     * Creates a Directory, using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param name
     *      location of the directory.
     * param flags
     *      the open mode.
     * return
     *      the directory instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static Directory createDirectory(String sagaFactoryClassname, URL name, int flags)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        Session session = SessionFactory.createSession(sagaFactoryClassname);
        return createDirectory(sagaFactoryClassname, session, name, flags);
    }


    /**
     * Creates a Directory for reading, using the default session.
     *
     * param name
     *      location of the directory.
     * return
     *      the directory instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static Directory createDirectory(URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createDirectory(name, Flags.READ.getValue());
    }


    /**
     * Creates a Directory for reading, using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param name
     *      location of the directory.
     * return
     *      the directory instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid file name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is not thrown, but a method may be invoked that may throw it (but
     *      not in this case).
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static Directory createDirectory(String sagaFactoryClassname, URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createDirectory(sagaFactoryClassname, 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<FileFactory, File> createFile(TaskMode mode,
            Session session, URL name, int flags)
            throws NotImplementedException, NoSuccessException {
        return createFile(null, mode, session, name, flags);
    }


    /**
     * Creates a task that creates a File.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<FileFactory, File> createFile(String sagaFactoryClassname, TaskMode mode,
            Session session, URL name, int flags)
            throws NotImplementedException, NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *            is thrown when the Saga factory could not be created.
     */
    public static Task<FileFactory, File> createFile(TaskMode mode,
            Session session, URL name) throws NotImplementedException,
            NoSuccessException {
        return createFile(mode, session, name, Flags.READ.getValue());
    }

    /**
     * Creates a task that creates a File for reading.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *           the task mode.
     * param session
     *           the session handle.
     * param name
     *           location of the file.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     * throws NoSuccessException
     *            is thrown when the Saga factory could not be created.
     */
    public static Task<FileFactory, File> createFile(String sagaFactoryClassname, TaskMode mode,
            Session session, URL name) throws NotImplementedException,
            NoSuccessException {
        return createFile(sagaFactoryClassname, mode, 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     * exception NoSuccessException
     *               is thrown when the default session could not be created or
     *               when the Saga factory could not be created.
     */
    public static Task<FileFactory, File> createFile(TaskMode mode, URL name,
            int flags) throws NotImplementedException, NoSuccessException {
        return createFile(mode, null, name, flags);
    }


    /**
     * Creates a task that creates a File, using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param name
     *            location of the file.
     * param flags
     *            the open mode.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *                when the Saga factory could not be created.
     */
    public static Task<FileFactory, File> createFile(String sagaFactoryClassname, TaskMode mode, URL n
            int flags) throws NotImplementedException, NoSuccessException {
        Session session = SessionFactory.createSession(sagaFactoryClassname);
        return createFile(sagaFactoryClassname, 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *                when the Saga factory could not be created.
     */
    public static Task<FileFactory, File> createFile(TaskMode mode, URL name)
            throws NotImplementedException, NoSuccessException {
        return createFile(mode, name, Flags.READ.getValue());
    }

    /**
     * Creates a task that creates a File for reading, using the default
     * session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param name
     *            location of the file.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *                when the Saga factory could not be created.
     */
    public static Task<FileFactory, File> createFile(String sagaFactoryClassname, TaskMode mode, URL n
            throws NotImplementedException, NoSuccessException {
        return createFile(sagaFactoryClassname, mode, 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<FileFactory, FileInputStream> createFileInputStream(
            TaskMode mode, Session session, URL name)
            throws NotImplementedException, NoSuccessException {
        return createFileInputStream(null, mode, session, name);
    }

    /**
     * Creates a task that creates a FileInputStream.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param session
     *            the session handle.
     * param name
     *            location of the file.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<FileFactory, FileInputStream> createFileInputStream(
            String sagaFactoryClassname, TaskMode mode, Session session, URL name)
            throws NotImplementedException, NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).doCreateFileInputStream(mode, session, name);
    }

    /**
     * Creates a task that creates a FileInputStream using the default session.
     *
     * param mode
     *            the task mode.
     * param name
     *            location of the file.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *                when the Saga factory could not be created.
     */
    public static Task<FileFactory, FileInputStream> createFileInputStream(
            TaskMode mode, URL name) throws NotImplementedException,
            NoSuccessException {

        return createFileInputStream(mode, null, name);
    }


    /**
     * Creates a task that creates a FileInputStream using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param name
     *            location of the file.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *                when the Saga factory could not be created.
     */
    public static Task<FileFactory, FileInputStream> createFileInputStream(
            String sagaFactoryClassname, TaskMode mode, URL name) throws NotImplementedException,
            NoSuccessException {
        return createFileInputStream(sagaFactoryClassname, mode, null, 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     */
    public static Task<FileFactory, FileOutputStream> createFileOutputStream(
            TaskMode mode, Session session, URL name)
            throws NotImplementedException, NoSuccessException {
        return createFileOutputStream(mode, session, name, false);
    }

    /**
     * Creates a task that creates a FileOutputStream.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param session
     *            the session handle.
     * param name
     *           location of the file.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     * throws NoSuccessException
     */
    public static Task<FileFactory, FileOutputStream> createFileOutputStream(
            String sagaFactoryClassname, TaskMode mode, Session session, URL name)
            throws NotImplementedException, NoSuccessException {
        return createFileOutputStream(sagaFactoryClassname, mode, session, name, false);
    }

    /**
     * Creates a task that creates a FileOutputStream.
     *
     * param mode
     *           the task mode.
     * param name
     *           location of the file.
     * param append
     *           when set, the file is opened for appending.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     * throws NoSuccessException
     *            is thrown when the Saga factory could not be created.
     */
    public static Task<FileFactory, FileOutputStream> createFileOutputStream(
            TaskMode mode, Session session, URL name, boolean append)
            throws NotImplementedException, NoSuccessException {
        return createFileOutputStream(null, mode, session, name, append);
    }


    /**
     * Creates a task that creates a FileOutputStream.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *           the task mode.
     * param name
     *           location of the file.
     * param append
     *           when set, the file is opened for appending.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<FileFactory, FileOutputStream> createFileOutputStream(
            String sagaFactoryClassname, TaskMode mode, Session session, URL name, boolean append)
            throws NotImplementedException, NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).doCreateFileOutputStream(mode, session, name, append);
    }

    /**
     * Creates a task that creates a FileOutputStream using the default session.
     *
     * param mode
     *            the task mode.
     * param name
     *            location of the file.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the default session could not be created or
     *             when the Saga factory could not be created.
     */
    public static Task<FileFactory, FileOutputStream> createFileOutputStream(
            TaskMode mode, URL name) throws NotImplementedException,
            NoSuccessException {
        return createFileOutputStream(mode, name, false);
    }

    /**
     * Creates a task that creates a FileOutputStream using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param name
     *            location of the file.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the default session could not be created or
     *             when the Saga factory could not be created.
     */
    public static Task<FileFactory, FileOutputStream> createFileOutputStream(
            String sagaFactoryClassname, TaskMode mode, URL name) throws NotImplementedException,
            NoSuccessException {
        return createFileOutputStream(sagaFactoryClassname, mode, name, false);
    }

    /**
     * Creates a task that creates a FileOutputStream.
     *
     * param mode
     *            the task mode.
     * param name
     *            location of the file.
     * param append
     *            when set, the file is opened for appending.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the default session could not be created or
     *             when the Saga factory could not be created.
     */
    public static Task<FileFactory, FileOutputStream> createFileOutputStream(
            TaskMode mode, URL name, boolean append)
            throws NotImplementedException, NoSuccessException {
        return createFileOutputStream(mode, null, name, append);
    }

    /**
     * Creates a task that creates a FileOutputStream.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param name
     *            location of the file.
     * param append
     *            when set, the file is opened for appending.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the default session could not be created or
     *             when the Saga factory could not be created.
     */
    public static Task<FileFactory, FileOutputStream> createFileOutputStream(
            String sagaFactoryClassname, TaskMode mode, URL name, boolean append)
            throws NotImplementedException, NoSuccessException {
        Session session = SessionFactory.createSession(sagaFactoryClassname);
        return createFileOutputStream(sagaFactoryClassname, 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<FileFactory, Directory> createDirectory(TaskMode mode,
            Session session, URL name, int flags)
            throws NotImplementedException, NoSuccessException {
        return createDirectory(null, mode, session, name, flags);
    }


    /**
     * Creates a task that creates a Directory.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<FileFactory, Directory> createDirectory(String sagaFactoryClassname, TaskMode m
            Session session, URL name, int flags)
            throws NotImplementedException, NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<FileFactory, Directory> createDirectory(TaskMode mode,
            Session session, URL name) throws NotImplementedException,
            NoSuccessException {
        return createDirectory(mode, session, name, Flags.READ.getValue());
    }

    /**
     * Creates a task that creates a Directory for reading.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param session
     *            the session handle.
     * param name
     *            location of the directory.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<FileFactory, Directory> createDirectory(String sagaFactoryClassname, TaskMode m
            Session session, URL name) throws NotImplementedException,
            NoSuccessException {
        return createDirectory(sagaFactoryClassname, 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *                when the Saga factory could not be created.
     */
    public static Task<FileFactory, Directory> createDirectory(TaskMode mode,
            URL name, int flags) throws NotImplementedException,
            NoSuccessException {
        return createDirectory(mode, null, name, flags);
    }


    /**
     * Creates a task that creates a Directory, using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param name
     *            location of the directory.
     * param flags
     *            the open mode.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *                when the Saga factory could not be created.
     */
    public static Task<FileFactory, Directory> createDirectory(String sagaFactoryClassname, TaskMode m
            URL name, int flags) throws NotImplementedException,
            NoSuccessException {
        Session session = SessionFactory.createSession(sagaFactoryClassname);
        return createDirectory(sagaFactoryClassname, 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *                when the Saga factory could not be created.
     */
    public static Task<FileFactory, Directory> createDirectory(TaskMode mode,
            URL name) throws NotImplementedException, NoSuccessException {
        return createDirectory(mode, name, Flags.READ.getValue());
    }


    /**
     * Creates a task that creates a Directory for reading, using the default
     * session.
     *
     * param sagaFactoryClassname
     *       the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param name
     *            location of the directory.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *                when the Saga factory could not be created.
     */
    public static Task<FileFactory, Directory> createDirectory(String sagaFactoryClassname, TaskMode m
            URL name) throws NotImplementedException, NoSuccessException {
```

```
        return createDirectory(sagaFactoryClassname, mode, name, Flags.READ.getValue());
    }
}
```

4.3.5 FileInputStream

```
package org.ogf.saga.file;

import java.io.InputStream;

import org.ogf.saga.SagaObject;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.task.Async;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;

/**
 * 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.
 */
public abstract class FileInputStream extends InputStream implements
        SagaObject, Async {

    /**
     * Clone is mentioned here because the inherited
     * {link java.lang.Object#clone()} cannot hide the public version in
     * {link SagaObject#clone()}.
     */
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }

    // Task versions of java.io.InputStream methods.

    /**
     * Creates a task that reads a byte from this stream.
     * See {link java.io.InputStream#read()}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<FileInputStream, Integer> read(TaskMode mode)
            throws NotImplementedException;
    /**
     * Creates task that reads (part of) a buffer from this stream.
     * See {link java.io.InputStream#read(byte[], int, int)}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<FileInputStream, Integer> read(TaskMode mode,
            byte[] buf, int off, int len) throws NotImplementedException;

    /**
     * Creates a task that reads a buffer from this stream.
     * See {link java.io.InputStream#read(byte[])}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public Task<FileInputStream, Integer> read(TaskMode mode, byte[] buf)
            throws NotImplementedException {
        return read(mode, buf, 0, buf.length);
    }

    /**
     * Creates a task that skips the specified number of bytes from this stream.
     * See {link java.io.InputStream#skip(long)}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<FileInputStream, Long> skip(TaskMode mode, long n)
            throws NotImplementedException;

    /**
     * Creates a task that determines how many bytes are available from this
     * stream.
     * See {link java.io.InputStream#available()}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<FileInputStream, Integer> available(TaskMode mode)
```

```
            throws NotImplementedException;

    /**
     * Creates a task that closes this stream.
     * See {link java.io.InputStream#close()}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<FileInputStream, Void> close(TaskMode mode)
            throws NotImplementedException;

    /**
     * Creates a task that marks the current position in this stream.
     * See {link java.io.InputStream#mark(int)}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<FileInputStream, Void> mark(TaskMode mode,
            int readlimit) throws NotImplementedException;

    /**
     * Creates a task that resets the position to the position last marked.
     * See {link java.io.InputStream#reset()}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<FileInputStream, Void> reset(TaskMode mode)
            throws NotImplementedException;

    /**
     * Creates a task that determines if {link java.io.InputStream#mark(int)}
     * is supported.  See {link java.io.InputStream#markSupported()}.
     *
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<FileInputStream, Boolean> markSupported(TaskMode mode)
            throws NotImplementedException;
```

```
}
```

4.3.6 FileOutputStream

```
package org.ogf.saga.file;

import java.io.OutputStream;

import org.ogf.saga.SagaObject;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.task.Async;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;

/**
 * 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.
 */
public abstract class FileOutputStream extends OutputStream implements
        SagaObject, Async {

    /**
     * Clone is mentioned here because the inherited
     * {link java.lang.Object#clone()} cannot hide the public version in
     * {link SagaObject#clone()}.
     */
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }

    // Task versions of java.io.OutputStream methods.

    /**
     * Creates a task that writes a byte to this stream.
     * See {link java.io.OutputStream#write(int)}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<FileOutputStream, Void> write(TaskMode mode, int b)
            throws NotImplementedException;

    /**
```

```
     * Creates a task that writes (part of) a buffer to this stream.
     * See {link java.io.OutputStream#write(byte[], int, int)}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<FileOutputStream, Void> write(TaskMode mode,
            byte[] buf, int off, int len) throws NotImplementedException;

    /**
     * Creates a task that writes a buffer to this stream.
     * See {link java.io.OutputStream#write(byte[])}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public Task<FileOutputStream, Void> write(TaskMode mode, byte[] buf)
            throws NotImplementedException {
        return write(mode, buf, 0, buf.length);
    }

    /**
     * Creates a task that flushes this stream.
     *  See {link java.io.OutputStream#flush()}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<FileOutputStream, Void> flush(TaskMode mode)
            throws NotImplementedException;

    /**
     * Creates a task that closes this stream.
     * See {link java.io.OutputStream#close()}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<FileOutputStream, Void> close(TaskMode mode)
            throws NotImplementedException;
}
```

4.3.7 SeekMode

```
package org.ogf.saga.file;

/**
 * Determines the seekmode for {link File#seek}: the specified offset is
 * interpreted with respect to the start, the current position, or the end.
 */
public enum SeekMode {
    /** Seek from the start of the file. */
    START(1),
    /** Seek from the current position in the file. */
    CURRENT(2),
    /** Seek from the end of the file. */
    END(3);

    private int value;

    private SeekMode(int value) {
        this.value = value;
    }

    /**
     * Returns the integer value of this enumeration literal.
     *
     * return the integer 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.

Some methods in the `LogicalDirectory` interface have been renamed slightly to avoid conflicts with methods in `NSDirectory`, as only the type of the return value differs, and Java forbids that.

4.4.1 LogicalDirectory

```
package org.ogf.saga.logicalfile;

import java.util.List;

import org.ogf.saga.attributes.AsyncAttributes;
import org.ogf.saga.error.AlreadyExistsException;
import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.DoesNotExistException;
import org.ogf.saga.error.IncorrectStateException;
import org.ogf.saga.error.IncorrectURLException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;
import org.ogf.saga.namespace.NSDirectory;
import org.ogf.saga.task.TaskMode;
import org.ogf.saga.task.Task;
import org.ogf.saga.url.URL;

/**
 * This interface represents a container for logical files in a logical file
 * name space.
 */
public interface LogicalDirectory extends NSDirectory,
        AsyncAttributes<LogicalDirectory> {

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL contains an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the LogicalDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception DoesNotExistException
     *      is thrown if the specified name does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public boolean isFile(URL name) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, DoesNotExistException,
            IncorrectStateException, TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when illegal flags are specified: only RECURSIVE
     *      (or NONE) is allowed, or one or more of the patterns is
     *      not correctly formatted.
     * exception IncorrectStateException
     *      is thrown when the LogicalDirectory is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public List<URL> find(String namePattern, String[] attrPattern, int flags)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException, TimeoutException,
            NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when one or more of the patterns is
     *      not correctly formatted.
     * exception IncorrectStateException
     *      is thrown when the LogicalDirectory is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public List<URL> find(String namePattern, String[] attrPattern)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException, TimeoutException,
            NoSuccessException;

    // 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL does not point to a directory,
     *      or is an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the LogicalDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public LogicalDirectory openLogicalDir(URL name, int flags)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Creates a new <code>LogicalDirectory</code> instance with read flag.
     *
     * param name
     *      directory to open.
     * return
     *      the opened directory instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL does not point to a directory,
     *      or is an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the LogicalDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      not thrown, but specified because a method may be invoked
     *      that can throw this exception, but will not in this case.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public LogicalDirectory openLogicalDir(URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Creates a new <code>LogicalFile</code> instance.
     *
     * param name
     *      logical file to open.
     * param flags
     *      defining the operation modus.
     * return
     *      the opened logical file.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL points to a directory,
     *      or is an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the LogicalDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public LogicalFile openLogicalFile(URL name, int flags)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Creates a new <code>LogicalFile</code> instance with read flag.
     *
     * param name
     *            logical file to open.
     * return the opened logical file.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL does not point to a directory,
     *      or is an invalid entry name.
     * exception IncorrectStateException
     *      is thrown when the LogicalDirectory is already closed.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      not thrown, but specified because a method may be invoked
     *      that can throw this exception, but will not in this case.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public LogicalFile openLogicalFile(URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    // 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<NSDirectory, Boolean> isFile(TaskMode mode, URL name)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<LogicalDirectory, List<URL>> find(TaskMode mode,
            String namePattern, String[] attrPattern, int flags)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<LogicalDirectory, List<URL>> find(TaskMode mode,
            String namePattern, String[] attrPattern)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<LogicalDirectory, LogicalDirectory> openLogicalDir(
            TaskMode mode, URL name, int flags) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<LogicalDirectory, LogicalDirectory> openLogicalDir(
            TaskMode mode, URL name) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<LogicalDirectory, LogicalFile> openLogicalFile(TaskMode mode,
            URL name, int flags) throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<LogicalDirectory, LogicalFile> openLogicalFile(TaskMode mode,
            URL name) throws NotImplementedException;
}
```

4.4.2 LogicalFile

```
package org.ogf.saga.logicalfile;

import java.util.List;

import org.ogf.saga.attributes.AsyncAttributes;
import org.ogf.saga.error.AlreadyExistsException;
import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.DoesNotExistException;
import org.ogf.saga.error.IncorrectStateException;
import org.ogf.saga.error.IncorrectURLException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;
import org.ogf.saga.namespace.NSEntry;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;
import org.ogf.saga.url.URL;

/**
 * A LogicalFile provides the means to handle the contents of logical files.
 */
public interface LogicalFile extends NSEntry, AsyncAttributes<LogicalFile> {

    /**
     * Adds a replica location to the replica set. Note: does never throw an
     * <code>AlreadyExists</code> exception!
     *
     * param name
     *      the location to add.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception IncorrectStateException
     *      is thrown when the logical file is closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void addLocation(URL name) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException, TimeoutException,
            NoSuccessException;

    /**
     * Removes a replica location from the replica set.
     *
     * param name
     *      the location to remove.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception IncorrectStateException
     *      is thrown when the logical file is closed.
     * exception DoesNotExistException
     *      is thrown when the logical file does not contain the specified URL.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void removeLocation(URL name) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Changes a replica location in the replica set.
     *
     * param nameOld
     *      the location to be updated.
     * param nameNew
     *      the updated location.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when either of the specified URLs is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception IncorrectStateException
     *      is thrown when the logical file is closed.
     * exception DoesNotExistException
     *      is thrown when the logical file does not contain the specified
     *      <code>nameOld</code> URL.
     * exception AlreadyExistsException
     *      is thrown when the logical file already contains the specified
     *      <code>nameNew</code> URL.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void updateLocation(URL nameOld, URL nameNew)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
```

```
     * Lists the locations in this location set.
     *
     * return
     *      the location list.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the logical file is closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public List<URL> listLocations() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or the file was opened ReadOnly or WriteOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>OVERWRITE</code> flag is not given.
     * exception DoesNotExistException
     *      is thrown if the target lies in a non-existing part of the
     *      name space, unless the CREATEPARENTS flag is given.
     * exception IncorrectStateException
     *      is thrown when the logical file is closed, or the location set
     *      is empty.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void replicate(URL name, int flags) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * Replicates a file from any of the known locations to a new location, with
     * default flags NONE.
     *
     * param name
     *      location to replicate to.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or the file was opened ReadOnly or WriteOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists.
     * exception DoesNotExistException
     *      is thrown if the target lies in a non-existing part of the
     *      name space.
      * exception IncorrectStateException
     *      is thrown when the logical file is closed, or the location set
     *      is empty.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void replicate(URL name) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    //
    // 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<LogicalFile, Void> addLocation(TaskMode mode, URL name)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<LogicalFile, Void> removeLocation(TaskMode mode, URL name)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<LogicalFile, Void> updateLocation(TaskMode mode, URL nameOld,
            URL nameNew) throws NotImplementedException;

    /**
     * Creates a task that lists the locations in this location set.
     *
     * param mode
     *            the task mode.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<LogicalFile, List<URL>> listLocations(TaskMode mode)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<LogicalFile, Void> replicate(TaskMode mode, URL name, int flags)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<LogicalFile, Void> replicate(TaskMode mode, URL name)
            throws NotImplementedException;
}
```

4.4.3 LogicalFileFactory

```
package org.ogf.saga.logicalfile;

import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.AlreadyExistsException;
import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.DoesNotExistException;
import org.ogf.saga.error.IncorrectURLException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;
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;
import org.ogf.saga.url.URL;

/**
 * Factory for objects from the logicalfile package.
 */
public abstract class LogicalFileFactory {

    private static LogicalFileFactory getFactory(String sagaFactoryName)
            throws NoSuccessException, NotImplementedException {
return ImplementationBootstrapLoader.getLogicalFileFactory(sagaFactoryName);
    }

    /**
     * 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 NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * 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 NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    protected abstract Task<LogicalFileFactory, LogicalFile> doCreateLogicalFile(
            TaskMode mode, Session session, URL name, int flags)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    protected abstract Task<LogicalFileFactory, LogicalDirectory> doCreateLogicalDirectory(
            TaskMode mode, Session session, URL name, int flags)
            throws NotImplementedException;
    /**
     * Creates a LogicalFile.
     *
     * param session
     *      the session handle.
     * param name
     *      location of the file.
     * param flags
     *      the open mode.
     * return
     *      the file instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static LogicalFile createLogicalFile(Session session, URL name,
            int flags) throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createLogicalFile((String) null, session, name, flags);
    }


    /**
     * Creates a LogicalFile.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param session
     *      the session handle.
     * param name
     *      location of the file.
     * param flags
     *      the open mode.
     * return
     *      the file instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static LogicalFile createLogicalFile(String sagaFactoryClassname, Session session, URL name
            int flags) throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).doCreateLogicalFile(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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is actually not thrown, but specified in the SAGA specifications.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static LogicalFile createLogicalFile(Session session, URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createLogicalFile(session, name, Flags.READ.getValue());
    }


    /**
     * Creates a LogicalFile using READ open mode.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param session
     *      the session handle.
     * param name
     *      location of the file.
     * return
     *      the file instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is actually not thrown, but specified in the SAGA specifications.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static LogicalFile createLogicalFile(String sagaFactoryClassname, Session session, URL name
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createLogicalFile(sagaFactoryClassname, 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static LogicalFile createLogicalFile(URL name, int flags)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createLogicalFile((Session) null, name, flags);
    }

    /**
     * Creates a LogicalFile using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param name
     *      location of the file.
     * param flags
     *      the open mode.
     * return
     *      the file instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static LogicalFile createLogicalFile(String sagaFactoryClassname, URL name, int flags)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createLogicalFile(sagaFactoryClassname, (Session) null, name, flags);
    }

    /**
     * Creates a LogicalFile using READ open mode, using the default session.
     *
     * param name
     *      location of the file.
     * return
     *      the file instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is actually not thrown, but specified in the SAGA specifications.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static LogicalFile createLogicalFile(URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createLogicalFile((Session) null, name);
    }

    /**
     * Creates a LogicalFile using READ open mode, using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param name
     *      location of the file.
     * return
     *      the file instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is actually not thrown, but specified in the SAGA specifications.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static LogicalFile createLogicalFile(String sagaFactoryClassname, URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createLogicalFile(sagaFactoryClassname, (Session) null, name);
    }


    /**
     * Creates a LogicalDirectory.
     *
     * param session
     *      the session handle.
     * param name
     *      location of the directory.
     * param flags
     *      the open mode.
     * return
     *      the directory instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static LogicalDirectory createLogicalDirectory(Session session,
            URL name, int flags) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException {
        return createLogicalDirectory((String) null, session, name, flags);
    }


    /**
     * Creates a LogicalDirectory.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param session
     *      the session handle.
     * param name
     *      location of the directory.
     * param flags
     *      the open mode.
     * return
     *      the directory instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static LogicalDirectory createLogicalDirectory(String sagaFactoryClassname, Session sessio
            URL name, int flags) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, AlreadyExistsException,
            DoesNotExistException, TimeoutException, NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).doCreateLogicalDirectory(session, name, flags);
    }

    /**
     * Creates a LogicalDirectory using READ open mode.
     *
     * param session
     *      the session handle.
     * param name
     *      location of the directory.
     * return
     *      the directory instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is actually not thrown, but specified in the SAGA specifications.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static LogicalDirectory createLogicalDirectory(Session session,
            URL name) throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createLogicalDirectory(session, name, Flags.READ.getValue());
    }

    /**
     * Creates a LogicalDirectory using READ open mode.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param session
     *      the session handle.
     * param name
     *      location of the directory.
     * return
     *      the directory instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is actually not thrown, but specified in the SAGA specifications.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static LogicalDirectory createLogicalDirectory(String sagaFactoryClassname, Session sessio
            URL name) throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createLogicalDirectory(sagaFactoryClassname, session, name, Flags.READ.getValue());
    }


    /**
     * Creates a LogicalDirectory using the default session.
     *
     * param name
     *      location of the directory.
     * param flags
     *      the open mode.
     * return
     *      the directory instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static LogicalDirectory createLogicalDirectory(URL name, int flags)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createLogicalDirectory((Session) null, name, flags);
    }

    /**
     * Creates a LogicalDirectory using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param name
     *      location of the directory.
     * param flags
     *      the open mode.
     * return
     *      the directory instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is thrown if the specified URL already exists, and the
     *      <code>CREATE</code> and <code>EXCLUSIVE</code> flags are given.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist, and the
     *      <code>CREATE</code> flag is not given.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static LogicalDirectory createLogicalDirectory(String sagaFactoryClassname, URL name, int f
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createLogicalDirectory(sagaFactoryClassname, (Session) null, name, flags);
    }


    /**
     * Creates a LogicalDirectory using READ open mode, using the default
     * session.
     *
     * param name
     *      location of the directory.
     * return
     *      the directory instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is actually not thrown, but specified in the SAGA specifications.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static LogicalDirectory createLogicalDirectory(URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createLogicalDirectory(name, Flags.READ.getValue());
    }

    /**
     * Creates a LogicalDirectory using READ open mode, using the default
     * session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param name
     *      location of the directory.
     * return
     *      the directory instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL is an invalid entry name.
     * exception IncorrectURLException
     *      is thrown when an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception AlreadyExistsException
     *      is actually not thrown, but specified in the SAGA specifications.
     * exception DoesNotExistException
     *      is thrown if the specified URL does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static LogicalDirectory createLogicalDirectory(String sagaFactoryClassname, URL name)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            AlreadyExistsException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createLogicalDirectory(sagaFactoryClassname, name, Flags.READ.getValue());
    }


    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<LogicalFileFactory, LogicalFile> createLogicalFile(
            TaskMode mode, Session session, URL name, int flags)
            throws NotImplementedException, NoSuccessException {
        return createLogicalFile((String) null, mode, session, name, flags);
    }

    /**
     * Creates a task that creates a LogicalFile.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<LogicalFileFactory, LogicalFile> createLogicalFile(
            String sagaFactoryClassname, TaskMode mode, Session session, URL name, int flags)
            throws NotImplementedException, NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).doCreateLogicalFile(mode, session, name, flags);
    }


    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<LogicalFileFactory, LogicalFile> createLogicalFile(
            TaskMode mode, Session session, URL name)
            throws NotImplementedException, NoSuccessException {
        return createLogicalFile(mode, session, name, Flags.READ.getValue());
    }

    /**
     * Creates a task that creates a LogicalFile using READ open mode.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param session
     *            the session handle.
     * param name
     *            location of the file.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<LogicalFileFactory, LogicalFile> createLogicalFile(
            String sagaFactoryClassname, TaskMode mode, Session session, URL name)
            throws NotImplementedException, NoSuccessException {
        return createLogicalFile(sagaFactoryClassname, mode, session, name, Flags.READ.getValue());
    }

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *                when the Saga factory could not be created.
     */
    public static Task<LogicalFileFactory, LogicalFile> createLogicalFile(
            TaskMode mode, URL name, int flags) throws NotImplementedException,
            NoSuccessException {
        return createLogicalFile(mode, (Session) null, name, flags);
    }

    /**
     * Creates a task that creates a LogicalFile using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param name
     *            location of the file.
     * param flags
     *            the open mode.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *                when the Saga factory could not be created.
     */
    public static Task<LogicalFileFactory, LogicalFile> createLogicalFile(
            String sagaFactoryClassname, TaskMode mode, URL name, int flags) throws NotImplementedExce
            NoSuccessException {
        return createLogicalFile(sagaFactoryClassname, mode, (Session) null, name, flags);
    }

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *                when the Saga factory could not be created.
     */
    public static Task<LogicalFileFactory, LogicalFile> createLogicalFile(
            TaskMode mode, URL name) throws NotImplementedException,
            NoSuccessException {
        return createLogicalFile(mode, name, Flags.READ.getValue());
    }
    /**
     * Creates a task that creates a LogicalFile using READ open mode, using the
     * default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param name
     *            location of the file.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *                when the Saga factory could not be created.
     */
    public static Task<LogicalFileFactory, LogicalFile> createLogicalFile(
            String sagaFactoryClassname, TaskMode mode, URL name) throws NotImplementedException,
            NoSuccessException {
        return createLogicalFile(sagaFactoryClassname, mode, name, Flags.READ.getValue());
    }


    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<LogicalFileFactory, LogicalDirectory> createLogicalDirectory(
            TaskMode mode, Session session, URL name, int flags)
            throws NotImplementedException, NoSuccessException {
        return createLogicalDirectory((String) null, mode, session, name, flags);
    }
    /**
     * Creates a task that creates a LogicalDirectory.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<LogicalFileFactory, LogicalDirectory> createLogicalDirectory(
            String sagaFactoryClassname, TaskMode mode, Session session, URL name, int flags)
            throws NotImplementedException, NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).doCreateLogicalDirectory(mode, session, name, flags);
    }

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<LogicalFileFactory, LogicalDirectory> createLogicalDirectory(
            TaskMode mode, Session session, URL name)
            throws NotImplementedException, NoSuccessException {
        return createLogicalDirectory(mode, session, name, Flags.READ.getValue());
    }
    /**
     * Creates a task that creates a LogicalDirectory using READ open mode.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *           the task mode.
     * param session
     *           the session handle.
     * param name
     *           location of the directory.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     * throws NoSuccessException
     *            is thrown when the Saga factory could not be created.
     */
    public static Task<LogicalFileFactory, LogicalDirectory> createLogicalDirectory(
            String sagaFactoryClassname, TaskMode mode, Session session, URL name)
            throws NotImplementedException, NoSuccessException {
        return createLogicalDirectory(sagaFactoryClassname, mode, session, name, Flags.READ.getValue()
    }

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     * exception NoSuccessException
     *               is thrown when the default session could not be created or
     *               when the Saga factory could not be created.
     */
    public static Task<LogicalFileFactory, LogicalDirectory> createLogicalDirectory(
            TaskMode mode, URL name, int flags) throws NotImplementedException,
            NoSuccessException {
        return createLogicalDirectory(mode, (Session) null, name, flags);
    }

    /**
     * Creates a task that creates a LogicalDirectory using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param name
     *            location of the directory.
     * param flags
     *            the open mode.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *                when the Saga factory could not be created.
     */
    public static Task<LogicalFileFactory, LogicalDirectory> createLogicalDirectory(
            String sagaFactoryClassname, TaskMode mode, URL name, int flags) throws NotImplementedExce
            NoSuccessException {
        return createLogicalDirectory(sagaFactoryClassname, mode, (Session) null, name, flags);
    }

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *                when the Saga factory could not be created.
     */
    public static Task<LogicalFileFactory, LogicalDirectory> createLogicalDirectory(
            TaskMode mode, URL name) throws NotImplementedException,
            NoSuccessException {
        return createLogicalDirectory(mode, name, Flags.READ.getValue());
    }

    /**
     * Creates a task that creates a LogicalDirectory using READ open mode,
     * using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param name
     *            location of the directory.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the default session could not be created or
     *                when the Saga factory could not be created.
     */
    public static Task<LogicalFileFactory, LogicalDirectory> createLogicalDirectory(
            String sagaFactoryClassname, TaskMode mode, URL name) throws NotImplementedException,
            NoSuccessException {
        return createLogicalDirectory(sagaFactoryClassname, mode, 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 `waitFor`, 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.

Java programmers are used to `java.io.InputStream` and `java.io.OutputStream`. Therefore, these are provided as `StreamInputStream` and `StreamOutputStream`, and methods are provided to obtain such streams from a `Stream` object.

4.5.1 Stream

```
package org.ogf.saga.stream;

import org.ogf.saga.SagaObject;
import org.ogf.saga.attributes.AsyncAttributes;
import org.ogf.saga.buffer.Buffer;
import org.ogf.saga.context.Context;
import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.IncorrectStateException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.SagaIOException;
import org.ogf.saga.error.TimeoutException;
import org.ogf.saga.monitoring.AsyncMonitorable;
import org.ogf.saga.task.Async;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;
import org.ogf.saga.url.URL;

/**
 * A client stream object.
 */
public interface Stream extends SagaObject, Async, AsyncAttributes<Stream>,
        AsyncMonitorable<Stream> {

    // 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 StreamServer#serve()} call, <code>null</code>
     * is returned.
     *
     * return
     *      the URL.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the stream is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public URL getUrl() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * Returns the remote authorization info. The returned context is
     * deep-copied.
     *
     * return
     *      the remote context.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the stream is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public Context getContext() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    // management methods

    /**
     * Establishes a connection to the target defined during the construction of
     * the stream.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the stream is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void connect() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * Establishes a connection to the target defined during the construction of
     * the stream.
     * param timeoutInSeconds
     *      the timeout in seconds.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the stream is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void connect(float timeoutInSeconds) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception IncorrectStateException
     *      is thrown when the stream is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public int waitFor(int what) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception IncorrectStateException
     *      is thrown when the stream is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public int waitFor(int what, float timeoutInSeconds)
            throws NotImplementedException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            IncorrectStateException, NoSuccessException;

    /**
     * Closes an active connection. This method performs a non-blocking close.
     * I/O is no longer possible. The stream is put in state CLOSED.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void close() throws NotImplementedException,
            NoSuccessException;

    /**
```

```
     * Closes an active connection. I/O is no longer possible. The stream is put
     * in state CLOSED.
     *
     * param timeoutInSeconds
     *      the timeout in seconds.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void close(float timeoutInSeconds) throws NotImplementedException,
            NoSuccessException;

    // 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 NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or if the file was opened WriteOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown if the provided buffer is not large enough to hold
     *      the required length.
     * exception IncorrectStateException
     *      is thrown when the Stream is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception SagaIOException
     *      is thrown on read failures when the SAGA specifications say that
     *      a POSIX error number should be returned (which does not really
     *      make sense for Java).

     */
    public int read(Buffer buffer, int len) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException,
            SagaIOException;

    /**
     * Reads a raw buffer from the stream.
     *
     * param buffer
     *      the buffer to store into.
     * return
     *      the number of bytes read.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or if the file was opened WriteOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown if the provided buffer is not large enough to hold
     *      the required length.
     * exception IncorrectStateException
     *      is thrown when the Stream is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception SagaIOException
     *      is thrown on read failures when the SAGA specifications say that
     *      a POSIX error number should be returned (which does not really
     *      make sense for Java).
     */
    public int read(Buffer buffer) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException,
            SagaIOException;

    /**
     * Obtains an InputStream from the stream.
     *
     * return
     *      the input stream.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or if the file was opened WriteOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the Stream is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception SagaIOException
     *      is thrown on read failures when the SAGA specifications say that
     *      a POSIX error number should be returned (which does not really
     *      make sense for Java).
     */
    public StreamInputStream getInputStream() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException, SagaIOException;

    /**
     * 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 NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or if the file was opened WriteOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown if the provided buffer is not large enough to hold
     *      the required (specified) length, or no buffer size is available.
     * exception IncorrectStateException
     *      is thrown when the Stream is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception SagaIOException
     *      is thrown on read failures when the SAGA specifications say that
     *      a POSIX error number should be returned (which does not really
     *      make sense for Java).
     */
    public int write(Buffer buffer, int len) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException,
            SagaIOException;
    /**
     * Writes a raw buffer to the stream.
     *
     * param buffer
     *      the data to be sent.
     * return
     *      the number of bytes written.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or if the file was opened WriteOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown if the provided buffer is not large enough to hold
     *      the required length, or bo buffer size is available.
     * exception IncorrectStateException
     *      is thrown when the Stream is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception SagaIOException
     *      is thrown on read failures when the SAGA specifications say that
     *      a POSIX error number should be returned (which does not really
     *      make sense for Java).
     */
    public int write(Buffer buffer) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            IncorrectStateException, TimeoutException, NoSuccessException,
            SagaIOException;

    /**
     * Obtains an OutputStream from the stream.
     *
     * return
     *      the output stream.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully, or if the file was opened WriteOnly.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the Stream is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     * exception SagaIOException
     *      is thrown on read failures when the SAGA specifications say that
     *      a POSIX error number should be returned (which does not really
     *      make sense for Java).
     */
    public StreamOutputStream getOutputStream() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException, SagaIOException;

    //
    // 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 StreamServer#serve()} call,
     * the URL will be <code>null</code>.
     *
     * param mode
     *            the task mode.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<Stream, URL> getUrl(TaskMode mode)
            throws NotImplementedException;

    /**
     * Creates a task that obtains the remote authorization info. The returned
     * context is deep-copied.
     *
     * param mode
     *            the task mode.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<Stream, Context> getContext(TaskMode mode)
            throws NotImplementedException;

    // 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<Stream, Void> connect(TaskMode mode)
            throws NotImplementedException;

    /**
     * Returns a task that establishes a connection to the target defined during
     * the construction of the stream.
     *
     * param mode
     *            the task mode.
     * param timeoutInSeconds
     *            the timeout in seconds.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<Stream, Void> connect(TaskMode mode, float timeoutInSeconds)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<Stream, Integer> waitFor(TaskMode mode, int what)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<Stream, Integer> waitFor(TaskMode mode, int what,
            float timeoutInSeconds) throws NotImplementedException;

    /**
     * Returns a task that closes an active connection.
     *
     * param mode
     *            the task mode.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<Stream, Void> close(TaskMode mode)
```

```
        throws NotImplementedException;

/**
 * Returns a task that closes an active connection.
 *
 * param mode
 *            the task mode.
 * param timeoutInSeconds
 *            the timeout in seconds.
 * return the task.
 * exception NotImplementedException
 *                is thrown when the task version of this method is not
 *                implemented.
 */
public Task<Stream, Void> close(TaskMode mode, float timeoutInSeconds)
        throws NotImplementedException;

// 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 NotImplementedException
 *                is thrown when the task version of this method is not
 *                implemented.
 */
public Task<Stream, Integer> read(TaskMode mode, Buffer buffer, int len)
        throws NotImplementedException;

/**
 * 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 NotImplementedException
 *                is thrown when the task version of this method is not
 *                implemented.
 */
public Task<Stream, Integer> read(TaskMode mode, Buffer buffer)
        throws NotImplementedException;
```

```
    /**
     * Creates a task that obtains an OutputStream from the stream.
     * param mode
     *               the task mode.
     * return the task.
     * exception NotImplementedException
     *                   is thrown when the task version of this method is not
     *                   implemented.
     */
    public Task<Stream, StreamInputStream> getInputStream(TaskMode mode)
            throws NotImplementedException;

    /**
     * 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 task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<Stream, Integer> write(TaskMode mode, Buffer buffer, int len)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<Stream, Integer> write(TaskMode mode, Buffer buffer)
            throws NotImplementedException;

    /**
     * Creates a task that obtains an OutputStream from the stream.
     * param mode
     *                the task mode.
     * return the task.
     * exception NotImplementedException
     *                    is thrown when the task version of this method is not
     *                    implemented.
     */
    public Task<Stream, StreamOutputStream> getOutputStream(TaskMode mode)
            throws NotImplementedException;
}
```

4.5.2 StreamServer

```
package org.ogf.saga.stream;

import org.ogf.saga.SagaObject;
import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.IncorrectStateException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;
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;
import org.ogf.saga.url.URL;

/**
 * A <code>StreamServer</code> object represents an endpoint for
 * a listening/server object that waits for client connections.
 */
public interface StreamServer extends SagaObject, Async,
        AsyncMonitorable<StreamServer>, Permissions<StreamServer> {

    // 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the stream server is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public URL getUrl() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * Waits for incoming client connections (like an accept of a serversocket).
     * The returned stream is in OPEN state. This call may block indefinitely.
     *
     * return
     *      the client connection.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the stream server is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public Stream serve() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * Establishes a connection to the stream server.
     * return
     *      the resulting stream.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the stream is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public Stream connect() throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * Establishes a connection to the stream server.
     * param timeoutInSeconds
     *      the timeout in seconds.
     * return
     *      the resulting stream.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the stream is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public Stream connect(float timeoutInSeconds) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * 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.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception IncorrectStateException
     *      is thrown when the stream server is already closed.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public Stream serve(float timeoutInSeconds) throws NotImplementedException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, IncorrectStateException,
            TimeoutException, NoSuccessException;

    /**
     * Closes a stream server. This is a non-blocking call.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void close() throws NotImplementedException, NoSuccessException;

    /**
     * Closes a stream server.
     *
     * param timeoutInSeconds
     *      the timeout in seconds.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public void close(float timeoutInSeconds) throws NotImplementedException,
            NoSuccessException;

    //
    // 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<StreamServer, URL> getUrl(TaskMode mode)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<StreamServer, Stream> serve(TaskMode mode)
            throws NotImplementedException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<StreamServer, Stream> serve(TaskMode mode,
            float timeoutInSeconds) throws NotImplementedException;

    /**
     * Returns a task that establishes a connection to the stream server.
     *
     * param mode
     *           the task mode.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<StreamServer, Stream> connect(TaskMode mode)
            throws NotImplementedException;

    /**
     * Returns a task that establishes a connection to the stream server.
     *
     * param mode
     *           the task mode.
     * param timeoutInSeconds
     *           the timeout in seconds.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<StreamServer, Stream> connect(TaskMode mode, float timeoutInSeconds)
            throws NotImplementedException;

    /**
     * Obtains a task that closes a stream server.
     *
     * param mode
     *           the task mode.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<StreamServer, Void> close(TaskMode mode)
            throws NotImplementedException;

    /**
     * Obtains a task that closes a stream server.
     *
     * param mode
     *           the task mode.
     * param timeoutInSeconds
     *            the timeout in seconds.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<StreamServer, Void> close(TaskMode mode, float timeoutInSeconds)
            throws NotImplementedException;
}
```

4.5.3 StreamFactory

```
package org.ogf.saga.stream;

import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.IncorrectURLException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;
import org.ogf.saga.session.Session;
import org.ogf.saga.session.SessionFactory;
import org.ogf.saga.task.TaskMode;
import org.ogf.saga.task.Task;
import org.ogf.saga.url.URL;
import org.ogf.saga.url.URLFactory;

/**
 * Factory for objects from the stream package.
 */
public abstract class StreamFactory {

    private static final String DEFAULT_SERVER_URL = "";

    private static StreamFactory getFactory(String sagaFactoryName)
    throws NoSuccessException, NotImplementedException {
return ImplementationBootstrapLoader.getStreamFactory(sagaFactoryName);
    }

    private static URL createDefaultURL() {
        URL url;
        try {
            url = URLFactory.createURL(DEFAULT_SERVER_URL);
        } catch (Throwable e) {
            // Cannot happen.
            throw new Error("Implementation error", e);

        }
        return url;
    }

    /**
     * Creates a Stream. To be provided by the implementation.
     *
     * param session
     *            the session handle.
     * param name
     *            location of the stream server.
     * return the stream.
     */
    protected abstract Stream doCreateStream(Session session, URL name)
            throws NotImplementedException, IncorrectURLException,
            BadParameterException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            TimeoutException, NoSuccessException;

    /**
     * Creates a StreamServer. To be provided by the implementation.
     *
     * param session
     *            the session handle.
     * param name
     *            location of the server.
     * return the stream server.
     */
    protected abstract StreamServer doCreateStreamServer(Session session,
            URL name) throws NotImplementedException, IncorrectURLException,
            BadParameterException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            TimeoutException, NoSuccessException;

    /**
     * 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 server.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    protected abstract Task<StreamFactory, Stream> doCreateStream(
            TaskMode mode, Session session, URL name)
            throws NotImplementedException;

    /**
     * Creates a task that creates a StreamServer. To be provided by the
     * implementation.
     *
     * param mode
     *           the task mode.
     * param session
     *           the session handle.
     * param name
     *           location of the server.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    protected abstract Task<StreamFactory, StreamServer> doCreateStreamServer(
            TaskMode mode, Session session, URL name)
            throws NotImplementedException;

    /**
     * Creates a Stream.
     *
     * param session
     *      the session handle.
     * param name
     *      location of the stream server.
     * return
     *      the stream.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL cannot be found.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static Stream createStream(Session session, URL name)
            throws NotImplementedException, IncorrectURLException,
            BadParameterException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            TimeoutException, NoSuccessException {
        return createStream((String) null, session, name);
    }

    /**
     * Creates a Stream.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param session
     *      the session handle.
     * param name
     *      location of the stream server.
     * return
     *      the stream.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL cannot be found.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static Stream createStream(String sagaFactoryClassname, Session session, URL name)
            throws NotImplementedException, IncorrectURLException,
            BadParameterException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            TimeoutException, NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).doCreateStream(session, name);
    }

    /**
     * Creates a Stream using the default session.
     *
     * param name
     *      location of the stream server.
     * return
     *      the stream.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL cannot be found.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static Stream createStream(URL name) throws NotImplementedException,
            IncorrectURLException, BadParameterException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, TimeoutException, NoSuccessException {
        return createStream((Session) null, name);
    }


    /**
     * Creates a Stream using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param name
     *      location of the stream server.
     * return
     *      the stream.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL cannot be found.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static Stream createStream(String sagaFactoryClassname, URL name) throws NotImplementedExce
            IncorrectURLException, BadParameterException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, TimeoutException, NoSuccessException {
        return createStream(sagaFactoryClassname, (Session) null, name);
    }


    /**
     * Creates a StreamServer.
     *
     * param session
     *      the session handle.
     * param name
     *      location of the server.
     * return
     *      the server.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL cannot be found.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
```

```
 * exception NoSuccessException
 *      is thrown when the operation was not successfully performed,
 *      and none of the other exceptions apply.
 */
public static StreamServer createStreamServer(Session session, URL name)
        throws NotImplementedException, IncorrectURLException,
        BadParameterException, AuthenticationFailedException,
        AuthorizationFailedException, PermissionDeniedException,
        TimeoutException, NoSuccessException {
    return createStreamServer((String) null, session, name);
}


/**
 * Creates a StreamServer.
 *
 * param sagaFactoryClassname
 *      the class name of the Saga factory to be used.
 * param session
 *      the session handle.
 * param name
 *      location of the server.
 * return
 *      the server.
 * exception NotImplementedException
 *      is thrown if the implementation does not provide an
 *      implementation of this method.
 * exception PermissionDeniedException
 *      is thrown when the method failed because the identity used did
 *      not have sufficient permissions to perform the operation
 *      successfully.
 * exception AuthorizationFailedException
 *      is thrown when none of the available contexts of the
 *      used session could be used for successful authorization.
 *      This error indicates that the resource could not be accessed
 *      at all, and not that an operation was not available due to
 *      restricted permissions.
 * exception AuthenticationFailedException
 *      is thrown when operation failed because none of the available
 *      session contexts could successfully be used for authentication.
 * exception TimeoutException
 *      is thrown when a remote operation did not complete successfully
 *      because the network communication or the remote service timed
 *      out.
 * exception BadParameterException
 *      is thrown when the specified URL cannot be found.
 * exception IncorrectURLException
 *      is thrown if an implementation cannot handle the specified
 *      protocol, or that access to the specified entity via the
 *      given protocol is impossible.
```

```
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static StreamServer createStreamServer(String sagaFactoryClassname, Session session, URL na
            throws NotImplementedException, IncorrectURLException,
            BadParameterException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            TimeoutException, NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).doCreateStreamServer(session, name);
    }

    /**
     * Creates a StreamServer.
     *
     * param session
     *      the session handle.
     * return
     *      the server.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL cannot be found.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static StreamServer createStreamServer(Session session)
            throws NotImplementedException, IncorrectURLException,
            BadParameterException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            TimeoutException, NoSuccessException {
        return createStreamServer(session, createDefaultURL());
    }

    /**
     * Creates a StreamServer.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param session
     *      the session handle.
     * return
     *      the server.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL cannot be found.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static StreamServer createStreamServer(String sagaFactoryClassname, Session session)
```

```
            throws NotImplementedException, IncorrectURLException,
            BadParameterException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            TimeoutException, NoSuccessException {
        return createStreamServer(sagaFactoryClassname, session, createDefaultURL());
    }


    /**
     * Creates a StreamServer using the default session.
     *
     * param name
     *      location of the stream server.
     * return
     *      the server.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL cannot be found.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static StreamServer createStreamServer(URL name)
            throws NotImplementedException, IncorrectURLException,
            BadParameterException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            TimeoutException, NoSuccessException {
```

```
        return createStreamServer((Session) null, name);
    }


    /**
     * Creates a StreamServer using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param name
     *      location of the stream server.
     * return
     *      the server.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL cannot be found.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static StreamServer createStreamServer(String sagaFactoryClassname, URL name)
            throws NotImplementedException, IncorrectURLException,
            BadParameterException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            TimeoutException, NoSuccessException {
        return createStreamServer(sagaFactoryClassname, (Session) null, name);
    }
    /**
     * Creates a StreamServer using the default session.
     *
     * return
     *      the server.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL cannot be found.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static StreamServer createStreamServer()
            throws NotImplementedException, IncorrectURLException,
            BadParameterException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            TimeoutException, NoSuccessException {
        return createStreamServer((Session) null);
    }


    /**
     * Creates a StreamServer using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * return
     *      the server.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL cannot be found.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static StreamServer createStreamServer(String sagaFactoryClassname)
            throws NotImplementedException, IncorrectURLException,
            BadParameterException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            TimeoutException, NoSuccessException {
        return createStreamServer(sagaFactoryClassname, (Session) null);
    }


    /**
     * Creates a task that creates a Stream.
     *
     * param mode
     *            the task mode.
     * param session
     *            the session handle.
     * param name
     *           location of the stream server.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<StreamFactory, Stream> createStream(TaskMode mode,
            Session session, URL name) throws NotImplementedException,
            NoSuccessException {
        return createStream(null, mode, session, name);
    }


    /**
     * Creates a task that creates a Stream.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *           the task mode.
     * param session
     *           the session handle.
     * param name
     *           location of the stream server.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<StreamFactory, Stream> createStream(String sagaFactoryClassname, TaskMode mode,
            Session session, URL name) throws NotImplementedException,
            NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).doCreateStream(mode, session, name);
    }

    /**
     * Creates a task that creates a Stream using the default session.
     *
     * param mode
     *           the task mode.
     * param name
     *           location of the stream server.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the Saga factory could not be created or
     *                when the default session could not be created.
     */
    public static Task<StreamFactory, Stream> createStream(TaskMode mode,
            URL name) throws NotImplementedException, NoSuccessException {
        return createStream(mode, null, name);
    }

    /**
     * Creates a task that creates a Stream using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param name
     *            location of the stream server.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the Saga factory could not be created or
     *                when the default session could not be created.
     */
    public static Task<StreamFactory, Stream> createStream(String sagaFactoryClassname, TaskMode mode,
            URL name) throws NotImplementedException, NoSuccessException {
        return createStream(sagaFactoryClassname, mode, null, name);
    }

    /**
     * Creates a task that creates a StreamServer.
     *
     * param mode
     *            the task mode.
     * param session
     *            the session handle.
     * param name
     *            location of the server.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<StreamFactory, StreamServer> createStreamServer(
            TaskMode mode, Session session, URL name)
            throws NotImplementedException, NoSuccessException {
        return createStreamServer(null, mode, session, name);
    }

    /**
     * Creates a task that creates a StreamServer.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param session
     *            the session handle.
     * param name
     *            location of the server.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<StreamFactory, StreamServer> createStreamServer(
            String sagaFactoryClassname, TaskMode mode, Session session, URL name)
            throws NotImplementedException, NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).doCreateStreamServer(mode, session, name);
    }


    /**
     * Creates a task that creates a StreamServer.
     *
     * param mode
     *            the task mode.
     * param session
     *            the session handle.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<StreamFactory, StreamServer> createStreamServer(
            TaskMode mode, Session session) throws NotImplementedException,
            NoSuccessException {
        return createStreamServer(mode, session, createDefaultURL());
    }


    /**
     * Creates a task that creates a StreamServer.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param session
     *            the session handle.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created.
     */
    public static Task<StreamFactory, StreamServer> createStreamServer(
            String sagaFactoryClassname, TaskMode mode, Session session) throws NotImplementedExceptic
            NoSuccessException {
        return createStreamServer(sagaFactoryClassname, mode, session, createDefaultURL());
    }

    /**
     * Creates a task that creates a StreamServer using the default session.
     *
     * param mode
     *            the task mode.
     * param name
     *            location of the server.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the Saga factory could not be created or
     *                when the default session could not be created.
     */
    public static Task<StreamFactory, StreamServer> createStreamServer(
            TaskMode mode, URL name) throws NotImplementedException,
            NoSuccessException {
        return createStreamServer(mode, null, name);
    }

    /**
     * Creates a task that creates a StreamServer using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * param name
     *            location of the server.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the Saga factory could not be created or
     *                when the default session could not be created.
     */
    public static Task<StreamFactory, StreamServer> createStreamServer(
            String sagaFactoryClassname, TaskMode mode, URL name) throws NotImplementedException,
            NoSuccessException {
        return createStreamServer(sagaFactoryClassname, mode, null, name);
    }

    /**
     * Creates a task that creates a StreamServer using the default session.
     *
     * param mode
     *            the task mode.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the Saga factory could not be created or
     *                when the default session could not be created.
     */
    public static Task<StreamFactory, StreamServer> createStreamServer(
            TaskMode mode) throws NotImplementedException, NoSuccessException {
        return createStreamServer(mode, createDefaultURL());
    }


    /**
     * Creates a task that creates a StreamServer using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *            the task mode.
     * return the task.
     * exception NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     * exception NoSuccessException
     *                is thrown when the Saga factory could not be created or
     *                when the default session could not be created.
     */
    public static Task<StreamFactory, StreamServer> createStreamServer(
            String sagaFactoryClassname, TaskMode mode) throws NotImplementedException, NoSuccessExcep
        return createStreamServer(sagaFactoryClassname, mode, createDefaultURL());
    }
}
```

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 {
    /** Data are available on the stream. */  READ(1),
    /** The stream is accepting data. */      WRITE(2),
    /** An error occured on the stream. */    EXCEPTION(4);

    private int value;

    private 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 {
    /** Initial state of a newly constructed stream. */
    NEW(1),
    /** State of a connected stream.                */
    OPEN(2),
    /**
     * State of a stream on which {link Stream#close()} was called.
     * This is a final state.
     */
    CLOSED(3),
    /** Remote party closed the connection. This is a final state. */
    DROPPED(4),
    /** An error occured on the stream. This is a final state. */
    ERROR(5);

    private int value;

    private StreamState(int value) {
        this.value = value;
    }

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

4.5.6 StreamInputStream

```
package org.ogf.saga.stream;

import org.ogf.saga.SagaObject;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.task.Async;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;

/**
 * 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. These streams can be obtained
 * through the
 * {link org.ogf.saga.stream.Stream#getInputStream() Stream.getInputStream()}
 * method.
 */
public abstract class StreamInputStream extends java.io.InputStream implements
```

```
        SagaObject, Async {

    /**
     * Clone is mentioned here because the inherited
     * {link java.lang.Object#clone()} cannot hide the public version in
     * {link SagaObject#clone()}.
     */
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }

    // Task versions of java.io.InputStream methods.

    /**
     * Creates a task that reads a byte from this stream.
     * See {link java.io.InputStream#read()}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<StreamInputStream, Integer> read(TaskMode mode)
            throws NotImplementedException;

    /**
     * Creates task that reads (part of) a buffer from this stream.
     * See {link java.io.InputStream#read(byte[], int, int)}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<StreamInputStream, Integer> read(TaskMode mode,
            byte[] buf, int off, int len) throws NotImplementedException;

    /**
     * Creates a task that reads a buffer from this stream.
     * See {link java.io.InputStream#read(byte[])}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public Task<StreamInputStream, Integer> read(TaskMode mode, byte[] buf)
            throws NotImplementedException {
        return read(mode, buf, 0, buf.length);
    }
    /**
     * Creates a task that skips the specified number of bytes from this stream.
     * See {link java.io.InputStream#skip(long)}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<StreamInputStream, Long> skip(TaskMode mode, long n)
            throws NotImplementedException;

    /**
     * Creates a task that determines how many bytes are available from this
     * stream.
     * See {link java.io.InputStream#available()}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<StreamInputStream, Integer> available(TaskMode mode)
            throws NotImplementedException;

    /**
     * Creates a task that closes this stream.
     * See {link java.io.InputStream#close()}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<StreamInputStream, Void> close(TaskMode mode)
            throws NotImplementedException;

    /**
     * Creates a task that marks the current position in this stream.
     * See {link java.io.InputStream#mark(int)}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<StreamInputStream, Void> mark(TaskMode mode,
            int readlimit) throws NotImplementedException;
    /**
     * Creates a task that resets the position to the position last marked.
     * See {link java.io.InputStream#reset()}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<StreamInputStream, Void> reset(TaskMode mode)
            throws NotImplementedException;

    /**
     * Creates a task that determines if {link java.io.InputStream#mark(int)}
     * is supported.
     * See {link java.io.InputStream#markSupported()}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<StreamInputStream, Boolean> markSupported(TaskMode mode)
            throws NotImplementedException;
}
```

4.5.7 StreamOutputStream

```
package org.ogf.saga.stream;

import org.ogf.saga.SagaObject;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.task.Async;
import org.ogf.saga.task.Task;
import org.ogf.saga.task.TaskMode;

/**
 * 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. These streams can be
 * obtained through the
 * {link org.ogf.saga.stream.Stream#getOutputStream() Stream.getOutputStream()}
 * method.
 */
public abstract class StreamOutputStream extends java.io.OutputStream implements
        SagaObject, Async {
```

```
    /**
     * Clone is mentioned here because the inherited
     * {link java.lang.Object#clone()} cannot hide the public version in
     * {link SagaObject#clone()}.
     */
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }

    // Task versions of java.io.OutputStream methods.

    /**
     * Creates a task that writes a byte to this stream.
     * See {link java.io.OutputStream#write(int)}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<StreamOutputStream, Void> write(TaskMode mode, int b)
            throws NotImplementedException;

    /**
     * Creates a task that writes (part of) a buffer to this stream.
     * See {link java.io.OutputStream#write(byte[], int, int)}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<StreamOutputStream, Void> write(TaskMode mode,
            byte[] buf, int off, int len) throws NotImplementedException;

    /**
     * Creates a task that writes a buffer to this stream.
     * See {link java.io.OutputStream#write(byte[])}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public Task<StreamOutputStream, Void> write(TaskMode mode, byte[] buf)
            throws NotImplementedException {
        return write(mode, buf, 0, buf.length);
    }
    /**
     * Creates a task that flushes this stream.
     * See {link java.io.OutputStream#flush()}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */
    public abstract Task<StreamOutputStream, Void> flush(TaskMode mode)
            throws NotImplementedException;

    /**
     * Creates a task that closes this stream.
     * See {link java.io.OutputStream#close()}.
     * param mode
     *      the task mode.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     */

    public abstract Task<StreamOutputStream, Void> close(TaskMode mode)
            throws NotImplementedException;
}
```

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.

In contrast to the language-independent SAGA specifications, the `Parameter` interface does not extend `Buffer` interface. The motivation for this is that most Java language bindings for RPC systems use `java.lang.Object` for parameters, implying that at least some types are serialized automatically. See,for instance, the Java bindings for Ninf-g and Apache XML-RPC.

4.6.1 RPC

```
package org.ogf.saga.rpc;

import org.ogf.saga.SagaObject;
import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.DoesNotExistException;
import org.ogf.saga.error.IncorrectStateException;
import org.ogf.saga.error.IncorrectURLException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;
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<RPC> {

    /**
     * Calls the remote procedure.
     *
     * param parameters
     *      arguments and results for the call.
     * exception IncorrectURLException
     *      may be thrown here because the RPC server that was
     *      specified to the factory may not have been contacted
     *      before invoking the call.
     * exception NoSuccessException
     *      is thrown for arbitrary backend failures, with a
     *      descriptive error message.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception BadParameterException
     *      is thrown when at least one of the parameters of the method
     *      call is ill-formed, invalid, out of bounds or otherwise not
     *      usable, or if the RPC server cannot be found.
     * exception DoesNotExistException
     *      is thrown when an operation cannot succeed because the RPC server
     *      does not recognize the specified method.
     * exception IncorrectStateException
     *      is thrown when the object is already closed.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     */
    public void call(Parameter... parameters) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, IncorrectStateException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * Non-blocking close of the RPC handle instance. Note for Java
     * implementations: A finalizer could be used in case the application
     * forgets to close.
     * exception NoSuccessException
     *      is thrown for arbitrary backend failures, with a
     *      descriptive error message.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception IncorrectStateException
     *      is in the SAGA specifications, but is not thrown when the object
     *      is already closed.
     */
    public void close() throws NotImplementedException,
            IncorrectStateException, NoSuccessException;

    /**
     * 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.
     * exception NoSuccessException
     *      is thrown for arbitrary backend failures, with a
     *      descriptive error message.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception IncorrectStateException
     *      is in the SAGA specifications, but is not thrown when the object
     *      is already closed.
     */
    public void close(float timeoutInSeconds) throws NotImplementedException,
            IncorrectStateException, NoSuccessException;

    //
    // 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 NotImplementedException
     *                is thrown when the task version of this method is not
     *                implemented.
     */
    public Task<RPC, Void> call(TaskMode mode, Parameter... parameters)
            throws NotImplementedException;

    /**
     * Creates a task for closing the RPC handle instance.
     *
     * param mode
     *           the task mode.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<RPC, Void> close(TaskMode mode) throws NotImplementedException;

    /**
     * Creates a task for closing the RPC handle instance.
     *
     * param mode
     *           the task mode.
     * param timeoutInSeconds
     *           seconds to wait.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    public Task<RPC, Void> close(TaskMode mode, float timeoutInSeconds)
            throws NotImplementedException;
}
```

4.6.2 Parameter

```
package org.ogf.saga.rpc;

import org.ogf.saga.SagaObject;

/**
 * Parameters for RPC calls. Unlike the language-independent SAGA specifications,
 * this interface does not extend Buffer, because existing Java language
 * bindings for RPC usually just use {link java.lang.Object}. See,
 * for instance, the Java bindings for Ninf-g and the Apache XML-RPC.
 */
public interface Parameter extends SagaObject {

    /**
     * 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();

    /**
     * Sets the parameter object.
     *
     * param object
     *            the parameter value.
     */
    public void setData(Object object);

    /**
     * Retrieves the current value of the parameter object.
     *
     * return the current parameter object value.
     */
    public Object getData();
}
```

4.6.3 RPCFactory

```
package org.ogf.saga.rpc;

import org.ogf.saga.bootstrap.ImplementationBootstrapLoader;
import org.ogf.saga.error.AuthenticationFailedException;
import org.ogf.saga.error.AuthorizationFailedException;
import org.ogf.saga.error.BadParameterException;
import org.ogf.saga.error.DoesNotExistException;
import org.ogf.saga.error.IncorrectURLException;
import org.ogf.saga.error.NoSuccessException;
import org.ogf.saga.error.NotImplementedException;
import org.ogf.saga.error.PermissionDeniedException;
import org.ogf.saga.error.TimeoutException;
import org.ogf.saga.session.Session;
import org.ogf.saga.session.SessionFactory;
import org.ogf.saga.task.TaskMode;
import org.ogf.saga.task.Task;
import org.ogf.saga.url.URL;

/**
 * Factory for objects from the RPC package.
 */
public abstract class RPCFactory {
    private static RPCFactory getFactory(String sagaFactoryName)
            throws NoSuccessException, NotImplementedException {
return ImplementationBootstrapLoader.getRPCFactory(sagaFactoryName);
    }

    /**
     * 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(Object data, IOMode mode)
            throws BadParameterException, NoSuccessException,
            NotImplementedException;

    /**
     * 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 NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, TimeoutException, NoSuccessException;

    /**
     * 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     */
    protected abstract Task<RPCFactory, RPC> doCreateRPC(TaskMode mode,
            Session session, URL funcname) throws NotImplementedException;
```

```
/**
 * 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.
 * exception NotImplementedException
 *      is thrown if the implementation does not provide an
 *      implementation of this method.
 * exception BadParameterException
 *      is thrown when the implementation cannot handle the given
 *      data object.
 * exception NoSuccessException
 *      is thrown if the implementation cannot create valid
 *      default values based on the available information.
 */
public static Parameter createParameter(Object data, IOMode mode)
        throws BadParameterException, NoSuccessException,
        NotImplementedException {
    return createParameter(null, data, mode);
}

/**
 * Creates a Parameter object. If the mode indicates an Out parameter,
 * <code>data</code> may be <code>null</code>.
 *
 * param sagaFactoryClassname
 *      the class name of the Saga factory to be used.
 * param data
 *      data to be used.
 * param mode
 *      IN, OUT, INOUT.
 * return
 *      the parameter.
 * exception NotImplementedException
 *      is thrown if the implementation does not provide an
 *      implementation of this method.
 * exception BadParameterException
 *      is thrown when the implementation cannot handle the given
 *      data object.
 * exception NoSuccessException
 *      is thrown if the implementation cannot create valid
 *      default values based on the available information.
 */
public static Parameter createParameter(String sagaFactoryClassname, Object data, IOMode mode)
```

```
            throws BadParameterException, NoSuccessException,
            NotImplementedException {
return getFactory(sagaFactoryClassname).doCreateParameter(data, mode);
    }

    /**
     * Creates a Parameter object. To be provided by the implementation.
     *
     * param mode
     *      IN, OUT, INOUT.
     * return
     *      the parameter.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception BadParameterException
     *      is not thrown, but this method may call a method that does
     *      throw it (although not in this case).
     * exception NoSuccessException
     *      is thrown if the implementation cannot create valid
     *      default values based on the available information.
     */
    public static Parameter createParameter(IOMode mode)
            throws BadParameterException, NoSuccessException,
            NotImplementedException {
        return createParameter(null, mode);
    }


    /**
     * Creates a Parameter object. To be provided by the implementation.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *      IN, OUT, INOUT.
     * return
     *      the parameter.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception BadParameterException
     *      is not thrown, but this method may call a method that does
     *      throw it (although not in this case).
     * exception NoSuccessException
     *      is thrown if the implementation cannot create valid
     *      default values based on the available information.
     */
    public static Parameter createParameter(String sagaFactoryClassname, IOMode mode)
            throws BadParameterException, NoSuccessException,
            NotImplementedException {
        return createParameter(sagaFactoryClassname, null, mode);
    }


    /**
     * Creates an IN Parameter object.
     *
     * param data
     *      data to be used.
     * return
     *      the parameter.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception BadParameterException
     *      is thrown when the implementation cannot handle the given
     *      data object.
     * exception NoSuccessException
     *      is thrown if the implementation cannot create valid
     *      default values based on the available information.
     */
    public static Parameter createParameter(Object data)
            throws BadParameterException, NoSuccessException,
            NotImplementedException {
        return createParameter(data, IOMode.IN);
    }


    /**
     * Creates an IN Parameter object.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param data
     *      data to be used.
     * return
     *      the parameter.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception BadParameterException
     *      is thrown when the implementation cannot handle the given
     *      data object.
     * exception NoSuccessException
     *      is thrown if the implementation cannot create valid
     *      default values based on the available information.
     */
    public static Parameter createParameter(String sagaFactoryClassname, Object data)
            throws BadParameterException, NoSuccessException,
            NotImplementedException {
        return createParameter(sagaFactoryClassname, data, IOMode.IN);
    }

    /**
     * Creates an IN Parameter object.
     *
     * return
     *      the parameter.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception BadParameterException
     *      is not thrown, but this method may call a method that does
     *      throw it (although not in this case).
     * exception NoSuccessException
     *      is thrown if the implementation cannot create valid
     *      default values based on the available information.
     */
    public static Parameter createParameter() throws BadParameterException,
            NoSuccessException, NotImplementedException {
        return createParameter(IOMode.IN);
    }

    /**
     * Creates an IN Parameter object.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * return
     *      the parameter.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception BadParameterException
     *      is not thrown, but this method may call a method that does
     *      throw it (although not in this case).
     * exception NoSuccessException
     *      is thrown if the implementation cannot create valid
     *      default values based on the available information.
     */
    public static Parameter createParameter(String sagaFactoryClassname) throws BadParameterException,
            NoSuccessException, NotImplementedException {
        return createParameter(sagaFactoryClassname, IOMode.IN);
    }

    /**
     * Creates a RPC handle instance.
     *
     * param session
     *      the session handle.
     * param funcname
     *      specification of the remote procedure.
     * return
     *      the RPC handle instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL cannot be found.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible. Note that this exception can also
     *      be thrown when actually invoking a {link RPC#call(Parameter...)},
     *      which may be more convenient for implementations.
     * exception DoesNotExistException
     *      is thrown if the specified function does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static RPC createRPC(Session session, URL funcname)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, TimeoutException, NoSuccessException {
        return createRPC((String) null, session, funcname);
    }


    /**
     * Creates a RPC handle instance.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param session
     *      the session handle.
     * param funcname
     *      specification of the remote procedure.
     * return
     *      the RPC handle instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL cannot be found.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible. Note that this exception can also
     *      be thrown when actually invoking a {link RPC#call(Parameter...)},
     *      which may be more convenient for implementations.
     * exception DoesNotExistException
     *      is thrown if the specified function does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static RPC createRPC(String sagaFactoryClassname, Session session, URL funcname)
            throws NotImplementedException, IncorrectURLException,
            AuthenticationFailedException, AuthorizationFailedException,
            PermissionDeniedException, BadParameterException,
            DoesNotExistException, TimeoutException, NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).doCreateRPC(session, funcname);
    }

    /**
     * Creates a RPC handle instance using the default session.
     *
     * param funcname
     *      specification of the remote procedure.
     * return
     *      the RPC handle instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL cannot be found.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible. Note that this exception can also
     *      be thrown when actually invoking a {link RPC#call(Parameter...)},
     *      which may be more convenient for implementations.
     * exception DoesNotExistException
     *      is thrown if the specified function does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static RPC createRPC(URL funcname) throws NotImplementedException,
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, DoesNotExistException, TimeoutException,
            NoSuccessException {
```

```
        return createRPC((Session) null, funcname);
    }

    /**
     * Creates a RPC handle instance using the default session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param funcname
     *      specification of the remote procedure.
     * return
     *      the RPC handle instance.
     * exception NotImplementedException
     *      is thrown if the implementation does not provide an
     *      implementation of this method.
     * exception PermissionDeniedException
     *      is thrown when the method failed because the identity used did
     *      not have sufficient permissions to perform the operation
     *      successfully.
     * exception AuthorizationFailedException
     *      is thrown when none of the available contexts of the
     *      used session could be used for successful authorization.
     *      This error indicates that the resource could not be accessed
     *      at all, and not that an operation was not available due to
     *      restricted permissions.
     * exception AuthenticationFailedException
     *      is thrown when operation failed because none of the available
     *      session contexts could successfully be used for authentication.
     * exception TimeoutException
     *      is thrown when a remote operation did not complete successfully
     *      because the network communication or the remote service timed
     *      out.
     * exception BadParameterException
     *      is thrown when the specified URL cannot be found.
     * exception IncorrectURLException
     *      is thrown if an implementation cannot handle the specified
     *      protocol, or that access to the specified entity via the
     *      given protocol is impossible. Note that this exception can also
     *      be thrown when actually invoking a {link RPC#call(Parameter...)},
     *      which may be more convenient for implementations.
     * exception DoesNotExistException
     *      is thrown if the specified function does not exist.
     * exception NoSuccessException
     *      is thrown when the operation was not successfully performed,
     *      and none of the other exceptions apply.
     */
    public static RPC createRPC(String sagaFactoryClassname, URL funcname) throws NotImplementedExcept
            IncorrectURLException, AuthenticationFailedException,
            AuthorizationFailedException, PermissionDeniedException,
            BadParameterException, DoesNotExistException, TimeoutException,
            NoSuccessException {
        return createRPC(sagaFactoryClassname, (Session) null, 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created or when
     *             the default session could not be created.
     */
    public static Task<RPCFactory, RPC> createRPC(TaskMode mode,
            Session session, URL funcname) throws NotImplementedException,
            NoSuccessException {
        return createRPC(null, mode, session, funcname);
    }


    /**
     * Creates a task that creates a RPC handle instance.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *           the task mode.
     * param session
     *           the session handle.
     * param funcname
     *           specification of the remote procedure.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     * throws NoSuccessException
     *             is thrown when the Saga factory could not be created or when
     *             the default session could not be created.
     */
    public static Task<RPCFactory, RPC> createRPC(String sagaFactoryClassname, TaskMode mode,
            Session session, URL funcname) throws NotImplementedException,
            NoSuccessException {
if (session == null) {
    session = SessionFactory.createSession(sagaFactoryClassname);
}
        return getFactory(sagaFactoryClassname).doCreateRPC(mode, 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 NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     * exception NoSuccessException
     *               is thrown when the Saga factory could not be created or
     *               when the default session could not be created.
     */
    public static Task<RPCFactory, RPC> createRPC(TaskMode mode, URL funcname)
            throws NotImplementedException, NoSuccessException {
        return createRPC(mode, null, funcname);
    }

    /**
     * Creates a task that creates a RPC handle instance using the default
     * session.
     *
     * param sagaFactoryClassname
     *      the class name of the Saga factory to be used.
     * param mode
     *           the task mode.
     * param funcname
     *           specification of the remote procedure.
     * return the task.
     * exception NotImplementedException
     *               is thrown when the task version of this method is not
     *               implemented.
     * exception NoSuccessException
     *               is thrown when the Saga factory could not be created or
     *               when the default session could not be created.
     */
    public static Task<RPCFactory, RPC> createRPC(String sagaFactoryClassname, TaskMode mode, URL func
            throws NotImplementedException, NoSuccessException {
        return createRPC(sagaFactoryClassname, mode, null, funcname);
    }
}
```

4.6.4 IOMode

```
package org.ogf.saga.rpc;

/**
 * Describes parameter modes.
 */
public enum IOMode {
    /** The parameter is an input parameter. */
    IN(1),
    /** The parameter is an output parameter. */
    OUT(2),
    /** The parameter is an input and output parameter. */
    INOUT(3);

    private int value;

    private IOMode(int value) {
        this.value = value;
    }

    /**
     * Returns the integer value of this enumeration literal.
     *
     * return the 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	**<u>Thilo Kielmann</u>**
`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), Daniel Templeton (Sun Microsystems), 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

5.3 Disclaimer

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.