GFD-R-P.90
SAGA-CORE-WG

Tom Goodale, Cardiff
Shantenu Jha, UCL[1]
Hartmut Kaiser, LSU
Thilo Kielmann, VU[1]
Pascal Kleijer, NEC
Andre Merzky, VU/LSU[1]
John Shalf, LBNL
Christopher Smith, Platform

Version: 1.0

December 31, 2007

A Simple API for Grid Applications (SAGA)

Status of This Document

This document provides information to the grid community, proposing *the core components for an extensible Simple API for Grid Applications (SAGA Core API)*. It is supposed to be used as input to the definition of language specific bindings for this API, and by implementors of these bindings. Distribution is unlimited.

Copyright Notice

Copyright © Open Grid Forum (*2007*). All Rights Reserved.

Abstract

This document specifies *the core components for the Simple API for Grid Applications (SAGA Core API)*, a high level, application-oriented API for grid application development. The scope of this API is derived from the requirements specified in GFD.71 ("A Requirements Analysis for a Simple API for Grid Applications"). It will in the future be extended by additional API extensions.

[1] editor

Contents

1 Introduction

This document specifies SAGA CORE, the Core of the *Simple API for Grid Applications*. SAGA is a high-level API that directly addresses the needs of application developers. The purpose of SAGA is two-fold:

1. Provide an **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.

1.1 How to read this Document

This document is an API *specification*, and as such targets *at 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. It will *very likely be insufficient*unlikely be sufficient to extract the embedded SIDL specification of the API *hope to and* implement a SAGA-compliant API. In particular, the general design considerations in Section 2 give essential, additional information to be taken into account for any implementation *to be considered in order to be* SAGA compliant.

This document is structured as follows. This Section **focusses** on the formal aspects of an OGF recommendation document. Section 2 outlines the general design considerations of the SAGA API. Sections 3 and 4 contain the SAGA API specification itself. Section 5 gives author contact information and provides disclaimers concerning intellectual property rights and copyright issues, according to OGF policies. Finally, Appendix A gives illustrative, non-normative, code examples of using the SAGA API.

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 [6].

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 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 General Design Considerations

This section *is addressing addresses* those aspects of the SAGA API specification XRepnthat are applicablecommon to most or all of the SAGA packages as defined in Sections 3 and 4.

2.1 API Scope and Design Process

The scope and requirements of the SAGA API have been defined by OGF's *Simple API for Grid Applications Research Group* (SAGA-RG). The SAGA-RG has collected as broad as possible a set of use cases which has been published as GFD.70 [17]. *From these use cases, the requirements on a SAGA API have been derived. The requirements analysis has been published as GFD.71 [18] The requirements for the SAGA API were derived from this use cases document, an analysis of which has been published as GFD.71 [18]. For the actual API definition (this document), the* SAGA-CORE Working Group *(SAGA-CORE-WG) has been established. The formal specification and resulting document is the work of the* SAGA-CORE Working Group *which was spawned from the SAGA-RG.*

2.1.1 Requirements from the SAGA Requirement Analysis

The SAGA Requirement Analysis [18] lists the following functional and non-functional requirements *on of* the SAGA API:

Functional Requirements

- Job submission and management should be supported by the SAGA API.
- Resource discovery should be supported by the SAGA API.
- Data management should be supported by the SAGA API.
- Efficient data access should be supported by the SAGA API.
- Data replication should be supported by the SAGA API.
- Persistent storage of application specific information should be supported by the SAGA API.
- Streaming of data should be supported by the SAGA API.
- Support for messages on top of the streaming API should be considered by the SAGA API.
- *Asynchronous notification should be supported by the SAGA API.*

- Asynchronous notification should be supported by the SAGA API.
- Application level event generation and delivery should be supported by the SAGA API.
- Application steering should be supported by the SAGA API, but more use cases would be useful.
- GridRPC should be supported by the SAGA API.
- Further communication schemes should be considered as additional use cases are submitted to the group.
- Access to data-bases does not currently require explicit support in the SAGA API.

Non-functional Requirements

- Asynchronous operations should be supported by the API.
- Bulk operations should be supported by the API.
- The *exception handling* of the API should allow for *application level* error recovery strategies.
- The SAGA API should be implementable on a variety of security infrastructures.
- The SAGA API should expose only a minimum of security details, if any at all.
- Auditing, logging and accounting should not be exposed in the API.
- Workflows do not require explicit support on API level.
- QoS does not require explicit support on API level.
- Transactions do not require explicit support ~~at the~~ *on* API level.

2.1.2 Requirement Adoption Strategy

The use cases expressed the above requirements different levels of importance or urgency. This reflects the fact that some functionality is considered more important or even vital (like file access and job submission) while other functionality is seen as "nice to have" by many use cases (like application steering). Also, the group of active people in the SAGA specification process constitutes a specific set of expertise and interest – and this set is, to some extent, reflected in the selection of SAGA packages specified in this document.

For example, as *we received* *there were* no use cases from the enterprise user community, *and also had no* *nor was there any* active participation from that community in the SAGA standardization process, no enterprise specific API package is included here *We need to provide at least one example of the such a package*. This does not imply that we consider them unnecessary, but rather reflects *our* *the* wish *and need* to *orient* *derive* the API on real use cases, and to avoid the creation of an API *for made up*from perceived use cases, and *from* half-baked expertise.

Scope of the SAGA API

As various sides expressed their need for the availablity of a useful (i.e. implementable and usable) API specification as quickly as possible, the SAGA-CORE-WG decided to follow a two-phase approach. The SAGA API, as described in this document, covers all requirements that are considered both urgent and sufficiently well understood to produce an API. Addressing the other *the less urgent or well understood* requirements is deferred to future versions, or extensions, of the SAGA API. Based upon this reasoning, areas of functionality (from now *on* referred to as *packages*) that are included in SAGA API are the following:

- jobs
- files (and logical files)
- streams
- *remote procedure calls [19]*
- *auxiliary API's for*
  - session handle and security context
  - asynchronous method calls (tasks)
  - access control lists
  - attributes
  - monitoring
  - error handling

Possible extensions to be included in future SAGA versions or extensions are:

- steering and extended monitoring
- possibly combining logical/physical files (read on logical files)
- persistent information storage (see, e.g. the GAT Advert Service [2])
- GridCPR [11]
- task dependencies (simple work flows and task batches)
- extensions to existing classes, based on new use cases

The packages as listed above do not imply a hierarchy of API interfaces: all packages are motivated by their use cases; there is no split into 'lower level' and 'higher level' packages. The only exception is the group of auxiliary **APIs**, which is considered orthogonal to the non-auxiliary *maybe use the word "primary" in lieu of non-auxilary; this also avoids the use of "core"* SAGA packages.

Dependencies between packages have been kept to a *minimal level* *minimum*, *so as* to allow each package to be used independently of any other; this *may* *will also* allow partially conformant API implementations (see below).

The term *CORE* in SAGA CORE refers to the fact that the scope of the API encompasses an initial required set of API objects and methods, which is **perceived** to be essential to the received use cases. *It is important to reiterate,* that the term, *again*, does not imply any hierarchy of API packages, such as CORE and SHELL packages etc. We will drop the use of *the* CORE when referring to the API and use the term in the context of the Working Group.

2.1.3 Relation to OGSA

The SAGA API specification effort has often been compared to, and seen as overlapping in scope and functionality to the OGSA standardization effort [10]. *This is NOT correct. Reasons are the following:* *This perceived overlap in scope and functionality is misleading for the following reasons:*

- OGSA applies to *the* service and middleware level.

  SAGA applies to *the* application level.

- OGSA aims at service and middleware developers.

  SAGA aims at application developers.

- OGSA is an architecture.

  SAGA is an API.

- OGSA strives to be complete, and to fully cover any potential **grid service** in its architectural frame.

  SAGA is by definition incomplete (80:20 rule), and aims *for coverage of* *to cover* the mostly used grid functionalities *on* *at the* application level. *, with NO ambition to be complete in any sense.*

- OGSA cannot sensibly interface to SAGA.

  SAGA implementations can interface to (a subset of) OGSA compliant services (and in fact usually will do so).

For these and more reasons we think that SAGA and OGSA are complementary, but by no means **competitive**. The only commonality we are aware of is the *broadness* *breadth* of both approaches: both OGSA and SAGA strive to cover more than one specific area of middleware and application functionality, respectively.

There have been discussions between the SAGA and OGSA groups *in* *of the* OGF, which tried to ensure that the SAGA specification does not imply any specific middleware properties, and in particular does not imply any state management which would contradict OGSA based middleware. Until now, we are not aware of any such conflict, and will continue to ensure seemless implementability on OGSA based middleware.

2.2 The SIDL Interface Definition Language

For the SAGA API, an object oriented (OO) approach was adopted, as it is easier to produce a procedural API from an OO API than the converse, and one of the goals of SAGA is to provide APIs which are as natural as possible in each implementation language. Advanced OO features such as polymorphism were avoided, both for simplicity and also to avoid complications when mapping to procedural languages.

The design team chose to use SIDL, the *Scientific Interface Definition Language* [4], for specifying the API. This provides a programming-language neutral represention of the API, but with well-defined syntax and clear mapping to implementation languages.

This document, however, slightly deviates from the original SIDL language definition. This section gives a brief introduction to SIDL, describes the respective deviations *we* used, and also contains a number of notes to implementors on how to interpret this specification.

SIDL, from the Babel project, is similar to COM and CORBA IDL, but has an emphasis on scientific computing, with support **for** multi-dimensional arrays, etc. Although the SAGA **specification** does not use these features extensively, the multilanguage scope of Babel for mappings from SIDL to programming languages appealed to the authors of this specification.

The key SIDL concepts used in this document are:

`package:`	specifies a name space (see note below)
`interface:`	set of methods
`class:`	stateful object and the associated set of methods
`method:`	service that can be invoked on a object
`type:`	constraint to value of method parameters

SIDL supports single inheritance of classes, and multiple inheritance of interfaces.

Method definitions have signatures, which define which parameters are accepted on method invocation. These parameters can be:

- `in`: input parameter, passed by value, assumed **constant**
- `out`: output parameter, passed by reference
- `inout`: input and output parameter, passed by reference

2.2.1 Deviations from SIDL in this Document

SIDL has the notion of packages, which are equivalent to Java packages or C++ name spaces. Packages are used in this specification, for the purpose of cross referencing different API sections. The packages are not *required* to show up in the **implementation's** class names or name spaces, apart from the top level 'saga' name space.

SIDL also has the notion of 'versions', which are actually required on packages. We do not use versions in this specification, as the specification itself is versioned, and we do not intend to introduce versioning on classes and interfaces.

SIDL allows **multi-dimensional** arrays, in the form `array<type,dim>`. As SAGA uses only one-dimensional arrays, this document uses the simplified notation `array<type>`.

SIDL defines a string to be a `char*`. We feel, however, that strings have more powerful and native expressions in some languages (such as C++, Perl and Java), and use `string` for these types. `char*`, conventionally used for binary inout memory chunks, is expressed in this document as `array<byte>`.

This specification defines all method calls as `void` (or rather does not specify any return type for method calls at all). Instead of explicit return values, we define `out` parameters, which are in SIDL parameters which are passed by reference. However, for this specification we expect language bindings to use the first specified output parameter as return value **of** function calls where appropriate, in particular for the synchronous versions of the function calls. The asynchronous versions will, by their very nature, stick to the `out` parameter scheme, as described in Section 3.10.

2.2.2 Default Parameter Values

This document, in several places, adds default values in the SIDL part of the API specification. It is up to the language bindings to exploit any native means for default parameter values. If this is not possible, the language binding CAN abstain from default parameter values. Also, if asynchronous method calls require additional parameters, which might affect the handling of default parameters in languages such as C and C++, the language binding CAN deviate from this document in that respect.

2.2.3 Constness

SIDL method parameters specified as `in` parameters are considered to be `const`, and MUST NOT be changed by the implementation. The SAGA language bindings SHOULD utilize language mechanisms to enforce constness of these parameters, if possible.

To our knowledge, SIDL does not allow the specification of constness **at** method level. This means, SIDL does not permit a specification of which methods must leave the state of the object unchanged. We considered the introduction of const modifiers, to achieve consistent semantics over different implementations. However, a short analysis of various implementation techniques convinced us that requiring method constness would raise significant limitations to SAGA implementors (e.g. for implementations with late binding), with no immediately visible advantage to SAGA users. Hence, we waived any method level constness requirements for now, but this topic might get picked up in future versions of the API, e.g. with respect to object serialization (which implies known and consistent object state *on* *at* serialization points).

2.2.4 Attributes and Metrics

The SIDL sections in this specification contain additional normative information which are inserted as SIDL comments. In particular these are definitions for *attributes* and *metrics*. *Format definitions and meaning for these entities and* specifications can be found in **Section 3.8** *"SAGA Attributes Interface"* and Section 3.9 *"SAGA Monitoring Model"*, respectively.

2.2.5 Method Specification Details

All methods defined in the SIDL specification sections are further explained in the `'Specification Details'` sections in this document. These details to

method specifications are *normative*. They are formatted as follows (example taken from the `saga::file` class):

```
    - read
      Purpose:  reads up to len_in bytes from the file into
                the buffer.
-     Format:   read               (in    array<byte> buffer,
+     Format:   read               (inout buffer      buf,
!                                   in    int         len_in = -1,
                                    out   int         len_out);
      Inputs:   len_in:            number of bytes to be read
!     InOuts:   buf:               buffer to read data into
      Outputs:  len_out:           number of bytes successfully
                                   read
+     PreCond:  -
+     PostCond: - the data from the file are available in the
+                 buffer.
+     Perms:    Read
      Throws:   NotImplemented
                BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
!     Notes:    - the actual number of bytes read into buffer
                  is returned in len_out.  It is not an error
                  to read less bytes than requested, or in fact
                  zero bytes, e.g. at the end of the file.
                - errors are indicated by returning negative
                  values for len_out, which correspond to
!                 negatives of the respective POSIX ERRNO error
                  code.
                - the file pointer is positioned at the end of
                  the byte area successfully read during this
                  call.
!               - the given buffer must be large enough to
!                 store up to len_in bytes, or managed by the
!                 implementation - otherwise a 'BadParameter'
!                 exception is thrown.
+               - the notes about memory management from the
+                 buffer class apply.
!               - if the file was opened in write-only mode (i.e.
!                 no 'Read' or 'ReadWrite' flag was given), this
!                 method throws an 'PermissionDenied' exception.
!               - if len_in is smaller than 0, or not given,
!                 the buffer size is used for len_in.
!                 If that is also not available, a
!                 'BadParameter' exception is thrown.
                - similar to read (2) as specified by POSIX
```

The following sections are used in these detailed specifications of class methods:

`Purpose:`	the aim of the method
`Format:`	the SIDL prototype of the method
`Inputs:`	descriptions of `in` parameters
`InOuts:`	descriptions of `inout` parameters
`Outputs:`	descriptions of `out` parameters
`PreCond:`	conditions for successful invocation
`PostCond:`	effects of successful invocation
`Perms:`	permissions required for the method
`Throws:`	list of exceptions the method can throw
`Notes:`	other details

`PreCond`'ition: an example for a precondition is a specific object state. *An implementation MUST check these Preconditions, and MUST refuse to execute the method if they are not met, and throw an exception accordingly.*

`PostCond`'tion: an example for a postcondition is a changed object state. *An implementation MUST ensure that the postconditions are met upon successful method invocation, and MUST flag an error otherwise.*

`Throws:` the exceptions listed in this section are the only SAGA exceptions which can be thrown by the method.

`Perms:` this section lists the permissions required to perform the method. If that permission is not available to the caller, a `PermissionDenied` exception MUST be thrown by the implementation.

`Notes:` can contain, for example, references to the origin and use of the method, conditions on which exceptions are to be raised, semantic details of invocations, consistency implications of invocations, and more. *These `Notes` are normative!*

2.2.6 Inheritance

The SAGA API specification limits class inheritance to *single inheritance* – a class can, nevertheless, implement multiple interfaces. Similar to the original SIDL syntax, this document uses the qualifiers `extends` to signal inheritance relations of a class, and `implements` to signal an interface to be provided by a class.

Almost all SAGA classes implement the `saga::object` interface (which provides, for example, a unique instance id and the `saga::error_handler` interface), but the classes usually implement several other interfaces as well.

For inherited classes and implemented **interfaces** holds: if methods are over-

loaded (i.e. redefined with the same name), the semantics of the overloaded methods *from the base class* still **apply** (i.e. all `Notes` given on the detailed method description apply). *That This does* also *hold holds* for `CONSTRUCTOR`s and `DESTRUCTOR`s, and also, for example, for a `close()` which **is implicitly** called on the base class' destruction.

2.2.7 The SAGA Interfaces

For some SAGA objects, such as for ***saga::logical_file****, SAGA interfaces, like the attribute interface, can allow access to remote entities. These methods should thus (a) also be availabe asynchronously, and (b) allow to apply the permission interface. However, asynchronous method calls and permissions make no sense for other, local SAGA objects, in particular on the SAGA Look-&-Feel level.*

Thus, instead of implementing the ***saga::async*** *and* ***saga::permissions*** *interface in the various interfaces in general, this specification defines that SAGA implementations MUST apply the following rules:*

- *SAGA classes and interfaces, which implement the* ***saga::async*** *interface, and thus implement the SAGA task model, MUST also implement that task model for the methods defined in the following interfaces:*
  - ***saga::attributes***
  - ***saga::permissions***
  - ***saga::monitorable***
  - ***saga::steerable***
- *SAGA classes and interfaces, which implement the* ***saga::permissions*** *interface, and thus implement the SAGA permission model, MUST also implement that permission model for the methods defined in the following interfaces:*
  - ***saga::attributes***
  - ***saga::monitorable***
  - ***saga::steerable***

2.3 Language Binding Issues

The abstract SAGA API specification, as provided by this document, is language independent, object oriented, and specified in SIDL. Normative bindings for specific languages, both object oriented and procedural, will be defined in additional documents.

This document contains several examples illustrating the use of the API, and these have naturally been shown in specific languages, such as C++. These examples should not be taken as normative, but merely as illustrative of the use of the API. When normative language bindings are available, these examples may be revised to reflect these bindings. In order to give an impression of the **Look-&-Feel** in other languages, Appendix A lists some of the examples in different languages. Again, Appendix A is illustrative, not normative.

Language bindings of the SAGA API shall provide the typical **Look-&-Feel** of the respective programming language. This comprises the syntax for the entitities (objects, methods, classes, etc.), but also, to some degree, **semantic** details for which it makes sense to vary them with the programming language. We summarize the **semantic details** here.

- In this document, flags are denoted as bitfields (specifically, integer enums which can be combined by logical AND and OR). **This** is for notational convenience, and a language binding should use the most natural mechanism available.
- Language bindings MAY want to express array style arguments as variable argument lists, if that is appropriate.
- This document specifies file lengths, buffer lengths and **offsets** as `int` types. We expect implementations to use **suitably** large native data types, and to stick to language specific types where possible (such as `size_t` for buffer lengths in `C`, and `off_t` for file lengths in `C`). The SAGA language bindings MUST include the types to be used by the implementations. In particular, 64 bit types SHOULD be used if they are available.
- The SAGA **attribute** interface defines attribute keys to be strings. The SAGA monitorable interface defines metric names to be strings. At the same time, many attributes and metrics are predefined in this specification. In order to avoid typos, and improve interoperability between multiple implementations, we expect language bindings to exploit native mechanisms to have these **predefined** attributes and metric names specified as literal constants. For example, in C/C++ we would expect the following defines for the stream package (amongst others):

```
#define SAGA_METRIC_STATE    "state"
#define SAGA_STREAM_NODELAY  "nodelay"
```

- *Language bindings MAY define additional constants for special parameter values. For example, in C/C++ we would expect the following defines for timeout values (amongst others):*

```
#define SAGA_WAIT_FOREVER    -1.0
#define SAGA_NOWAIT           0.0
```

- Object lifetime management may be language specific. See Section 2.5.3.

- Concurrency control may be language specific. See Section 2.6.4.
- Thread safety may be language specific. See Section 2.6.5.

2.4 Compliant Implementations

A SAGA implementation MUST follow the SAGA API specification, and the language binding(s) for its respective programming language(s), both syntactically and semantically. *With respect to syntax, the language binding documents overrule this document, in case of contradictions.* This means that any method MUST be implemented with the syntax and with the semantics specified in this document *and the applicable language bindings*, or not be implemented at all (i.e. MUST then throw the `NotImplemented` exception).

The `NotImplemented` exception MUST, however, be used only in necessary cases, for example if an underlying **grid** middleware does not provide some capability, and if this capability can also not be emulated. The implementation MUST carefully document and motivate the use of the `NotImplemented` exception.

An implementation of the SAGA API is *"SAGA compliant"* if it implements all objects and methods of the SAGA API specification, possibly using the `NotImplemented` exception, as outlined above.

An implementation of the SAGA API is *"partially SAGA compliant"* if it implements only some packages, but implements those completely. It is, as with compliant implementations, acceptable to have methods that are not implemented at all (and thus throw a `NotImplemented` error).

All other implementations of the SAGA API are *"not SAGA compliant"*.

The SAGA **Look-&-Feel** classes and interfaces *(see Section 3)* (`exception`, `error_handler`, `object`, `session`, `context`, *`permissions`, `buffer`,* `attributes`, `callback`, `metric`, `monitorable`, `steerable`, `async`, `task`, and `task_container`) MUST be implemented completely for an implementation to be compliant. A partially compliant implementation MUST implement those SAGA **Look-&-Feel** classes and interfaces which are used by the packages the implementation **intends** to provide. A method in the SAGA **Look-&-Feel** classes and interfaces MUST NOT throw the `NotImplemented` exception.

Note that the exposure of additional (e.g. backend specific) classes, methods, or attributes within the SAGA API (e.g. within the `saga` name space) is considered to *break SAGA compliance*, unless *explicitly* allowed by this specification, as such extensions would bind applications to this specific implementation, and limit their portability, *which is the latter being* a declared goal of the SAGA approach.

The SAGA CORE Working Group will strive to provide, along with the language binding documents, complicance tests for implementors. It should also be noted that the SAGA language binding documents MAY specify deviations from the API syntax and semantics specified in this documents. In this case, the language binding specification supersedes this language independent specification. The language binding specifications MUST *, however,* strive to keep the set of differences to this specification as small as possible.

2.4.1 Early versus late binding

An implementation may choose to use late binding to middleware. This means that the middleware binding might change between subsequent SAGA calls. For example, a `file.open()` might be performed via the HTTP binding, but a subsequent `read()` on this file might *fail, and instead* be performed with GridFTP.

Late binding has some advantages in terms of flexibility and error recovery. However, it implies a certain amount of object state to be kept on client side, which might have semantic consequences. For example, a `read()` operation might fail on HTTP for some reasons, but might succeed via GridFTP. The situation might be reversed for `write()`. In order to allow alternating access via both protocols, the file pointer information (e.g. the file object state) must be held on client side.

It is left to a later experience document about the SAGA API implementations to discuss potential problems arising from early/late binding implementations, with respect to semantic conformance to the SAGA API specification. It should be noted here that method-level constness would represent a major obstacle for late binding implementations.

Late binding MUST NOT delay the check of error conditions if this is semantically required by the specification. For example, a `file.open()` should check for the existence of the file, even if the implementation may bind to a different middleware on subsequent operations on this file.

2.5 Object Management

The API specification in Sections 3 and 4 defines various kinds of objects. Here, we describe generic design considerations about managing these objects.

2.5.1 Session Management

The specification introduces a `saga::session` object, which acts as session handle. A session thereby identifies objects and operations which are sharing information, such as security details. *Also*, objects and methods from different sessions *are guaranteed to not to* *MUST NOT* share any information. *, and are completely shielded from each other.* This will allow *an* application to communicate with different **grids** and VOs at the same time, or to assume different IDs at the same time. Many applications, however, will have no need for explicit session handling. For those cases, a default **SAGA** session is used if no explicit `saga::session` object is created and used.

Any SAGA object is associated with a session at creation time, by using the respective `saga::session` instance as first argument to the constructor. If the session argument is **omitted**, the object is associated with the default session. SAGA objects created from other SAGA objects (such as a `saga::file` instance created by calling `open()` on a `saga::directory` instance) inherit the **parent's** session. The remainder of the document refers to the default session instance as `theSession`.

A `saga::context` instance is used to encapsulate a virtual identity, such as a Globus certificate or an ssh key pair. Multiple context instances can be associated with one session, and only **that** context information MUST be used to perform any operation in this session (i.e. on objects associated with this session). If no `saga::context` instances are explicitly added to a SAGA session, the SAGA implementation MAY associate one or more default contexts with any new session, including the default session. In fact, the default session can ONLY use these default contexts.

2.5.2 Shallow versus Deep Copy

Copy operations **on** SAGA objects are, by default, shallow. This applies, for example, when SAGA objects are passed by value, or by assignment operations. Shallow copy means that the orginal object instance and the new (copied) instance share state. For example, the following code snippet

Code Example

```
1  saga::file f1 (url);        // file pointer is at 0
2  saga::file f2 = f1;         // shallow copy
3
4  cout << "f1 is at " << f1.seek (0, Current) << "\n";
5  cout << "f2 is at " << f2.seek (0, Current) << "\n";
6
7  f1.seek (10, Current);      // change state
8
```

```
9    cout << "f1 is at " << f1.seek (0, Current) << "\n";
10   cout << "f2 is at " << f2.seek (0, Current) << "\n";
```

would yield the following output (comments added):

```
   f1 is at 0
   f2 is at 0   -> shallow copy of f1

   f1 is at 10  -> state of f1 changes
   f2 is at 10  -> state of f2 changes too: it is shared
```

The SAGA API allows, however, to perform deep copies on all SAGA objects, by explicitly using the `clone()` method. The changed code snippet:

Code Example

```
1    saga::file f1 (url);        // file pointer is at 0
2    saga::file f2 = f1.clone(); // deep copy
3
4    cout << "f1 is at " << f1.seek (0, Current) << "\n";
5    cout << "f2 is at " << f2.seek (0, Current) << "\n";
6
7    f1.seek (10, Current);      // change state
8
9    cout << "f1 is at " << f1.seek (0, Current) << "\n";
10   cout << "f2 is at " << f2.seek (0, Current) << "\n";
```

would then yield the following output (comments added):

```
   f1 is at 0
   f2 is at 0   -> deep copy of f1

   f1 is at 10  -> state of f1 changes
!  f2 is at 0   -> state of f2 did not change, it is not shared
```

SAGA language bindings MAY deviate from these semantics if (and only if) these semantics would be non-intuitive in the target language.

If a SAGA object gets (deeply) copied by the `clone` method, its complete state is copied, with the exception of:

- information about previous error conditions (*is not copied,* see Section 3.1),
- callbacks on metrics (*are not copied,* see Section 3.9).
- *the session the object was created in (is shallow copied, see Section 3.5)*,

Not copying previous error conditions disambiguates error handling. *Not copying the session ensures that the same session is continued to be shared between objects in that session, as intended.* Not copying registered callbacks is required to ensure proper functioning of the callback invocation mechanism, as callbacks have an inherent mechanism to allow callbacks to be called *exactly* once. Copying callbacks would undermine that mechanism, as callbacks could be called more than once (once on the original metric, once on the copied metric).

Note that a copied object will, in general, point to the same remote instance. For example, the copy of a `saga::job` instance will not cause the spawning of a new remote job, but will merely create a new handle to the same remote process the first instance pointed to. The new object instance is just a new handle which is in the same state as the original handle – from then on, the two handles have a life of their own. Obviously, operations on one SAGA object instance may still in fact influence the copied instance, e.g. if `cancel()` is called on either one.

Note also, that the deep/shallow copy semantics is the same for synchronous and asynchronous versions of any SAGA method call.

2.5.3 Object State Lifetime

In general, the lifetime of SAGA object instances is defined as natively expected in the respective languages, so **it** is usually explicitly managed, or implicitly defined by scoping, or in some languages implicitly managed by garbage collection mechanisms.

The SAGA API semantics, in particular asynchronous operations, tasks, and monitoring metrics require, however, that the state of certain objects must be able to survive the lifetime of the context in which they **were** created. As state in these situations is shared with the original object instance, this may imply in some languages that the respective objects must survive as well.

In particular, object state MUST be available in the following situations:

- The state of a `saga::object` instance *MUST* be available to all tasks

created on this object instance.

- The state of a `saga::object` instance *MUST* be available to all metrics created on this object instance.
- The state of a `saga::session` instance *MUST* be available to all objects created in this session.
- The state of a `saga::context` instance *MUST* be available to all sessions this context instance was added to.
- *The state of the default session MUST be available to the first invocation of any SAGA API method, and SHOULD be available for the remaining lifetime of the SAGA application.*

Due to the diversity of lifetime management used in existing programming languages, this document can not prescribe a single mechanism to implement objects or object states that survive the context they were created in. It is subject to individual language binding documents to prescribe such mechanisms, and to define responsibilities for object creation and destruction, both for SAGA implementations and for application programs, in order to match requirements and common-sense in the respective languages.

The SAGA specification implies that object state is shared in the following situations:

- **an** asynchronous operation is invoked on an object, creating a task instance;
- a SAGA object is passed as argument to a (synchronous or asynchronous) method call.

Those method calls that deviate from these semantics denote **this** in their `PostCond`'itions (e.g. prescribe that a deep copy of state occurs).

2.5.4 Freeing of Resources and Garbage Collection

The destruction of objects in distributed systems has its own subtle problems, as has the interruption of remote operations. In particular it cannot be assumed that a destructor can both return timely *and* ensure the de-allocation of all (local and remote) resources. In particular, as a remote connection breaks, no guarantees whatsoever can be made about the de-allocation of remote resources.

In particular for SAGA tasks, which represent asynchronous remote operations, we expect implementations to run into this problem space, for example

if `cancel()` is invoked on this task. To have common semantic guidelines for resource de-allocation, we define:

1. On explicit or implicit object destruction, and on explicit or implicit interruption of synchronous and asynchronous method invocations, SAGA implementations MUST make a best-effort attempt to free associated resources immediately[1].
2. If the immediate de-allocation of resources is not possible, for whichever reasons, the *respective interrupting or destructing* methods MUST return immediately, but the resource de-allocation MAY be delayed indefinitely. However, as of (1), the best effort strategy to free these resources eventually MUST stay in place.
3. Methods whose semantics depend on successful or unsuccessful de-allocation of resources (such as `task.cancel()` or `file.close()`) allow for an optional `float` argument, which defines a timeout for this operation *(see Section 2.6.3)*. If resource de-allocation does not succeed within this timeout period, a `NoSuccess` exception MUST be thrown. Negative values imply to wait forever, a . *A* value of zero (the default) implies that the method can return immediately*; no exception is thrown*, even if some resources could not be de-allocated. In any case, the best-effort policy as described above applies.

SAGA implementations MUST motivate and document any deviation from this behaviour. See also Section 2.4 on compliant implementations.

2.5.5 Destructors and `close()`

Destructors are implying a call to `close()` of the respective object (if a `close()` is defined for that class), unless, as described above, tasks are still using the respective resources *– then the close is delayed until the last of these tasks is destroyed (see 2.5.3)*. It must be noted that, unlike when using a direct call to `close()`, exceptions occuring on such an implicit `close()` cannot be communicated to the application: throwing exceptions in destructors is, in general, considered unclean design, and is in many languages outright forbidden. Thus, an explicit `close()` should be used by the application if feedback about eventual error conditions are *is* required. Otherwise, an implicit `close()` on object destruction will silently discard such error conditions (exceptions).

[1] *Immediately* in the description above means: within the expected response time of the overall system, but not longer.

2.6 Asynchronous Operations and Concurrency

In this section, we describe the general design considerations related to asynchronous operations, concurrency control, and multithreading.

2.6.1 Asynchronous Function Calls

The need for asynchronous calls was explicitly stated by the use cases, as reasonable synchronous behaviour cannot always be expected from grids. The SAGA task interface allows the creation of an asynchronous version of each SAGA API method call. The SIDL specification lists only the synchronous version of the API methods, but all *classes* implementing the task interface MUST provide the various asynchronous methods as well. Please see Section 3.10 for details on the task interface.

2.6.2 Asynchronous Notification

Related to this topic, the group also discussed the merits of callback and polling mechanisms and agreed that a callback mechanism should be used in SAGA to allow for asynchronous notification. In particular, this mechanism should allow for notification on the completion of asynchronous operations, i.e. task state changes. However, polling for states and other events is also supported.

2.6.3 Timeouts

Several methods in the SAGA API support the synchronization of concurrent operations. Often, those methods accept a `float` timeout parameter. The semantics of *this* parameter*s* *MUST be* as follows:

`timeout < 0.0` – wait forever
`timeout = 0.0` – return immediately
`timeout > 0.0` – wait for this many seconds

These methods *MUST not* cause a `Timeout` exception as the timeout period passes, but *MUST* return silently. For a description of the `Timeout` exception, see Section 3.1.

The various methods often define *different* default timeouts. For timeouts on `close()` methods, the description of resource de-allocation policies in Section 2.5.4 is also relevant.

2.6.4 Concurrency Control

Although limited, SAGA defines a de-facto concurrent programming model, via the task model and the asynchronous notification mechanism. Sharing of object state among concurrent units (e.g. tasks) is intentional and necessary for addressing the needs of various use cases. Concurrent use of shared state, however, requires concurrency control to avoid unpredictable behavior.

(Un)fortunately, a large variety of concurrency control mechanisms exist, with different programming languages lending themselves to certain flavors, like object locks and monitors in Java, or POSIX mutexes in C-like languages. For some use cases of SAGA, enforced concurrency control mechanisms might be both unnecessary and counter productive, leading to increased programming complexity and runtime overheads.

Because of these constraints, SAGA does not enforce concurrency *control* mechanisms on its implementations. Instead, it is the responsibility of the application programmer to ensure that her program will execute correctly in all possible orderings and interleavings of the concurrent units. The application programmer is free to use any concurrency control scheme (like locks, mutexes, or monitors) in addition to the SAGA API.

2.6.5 Thread Safety

We expect implementations of the SAGA API to be thread safe. Otherwise, the SAGA task model would be difficult to implement, and would also be close to useless. However, we acknowledge that specific languages might have trouble with (a) expressing the task model as it stands, and (b) might actually be successful to implement the API single threaded, and non-thread safe. Hence, we expect the language bindings to define if compliant implementations in this language MUST or CAN be thread safe – with MUST being the default, and CAN requiring good motivation.

2.7 State Diagrams

Several objects in SAGA have a *state* attribute or metric, which implies a state diagram for these objects. That means, that instances of these objects can undergo well defined state transitions, which are either triggered by calling specific methods on these object instances, or by calling methods on other object instances affecting these instances, or are triggered by internal events, for example by backend activities. State diagrams as shown in Figure 1 are used to define the available states, and the **allowed** state transitions. These diagrams are *normative*.

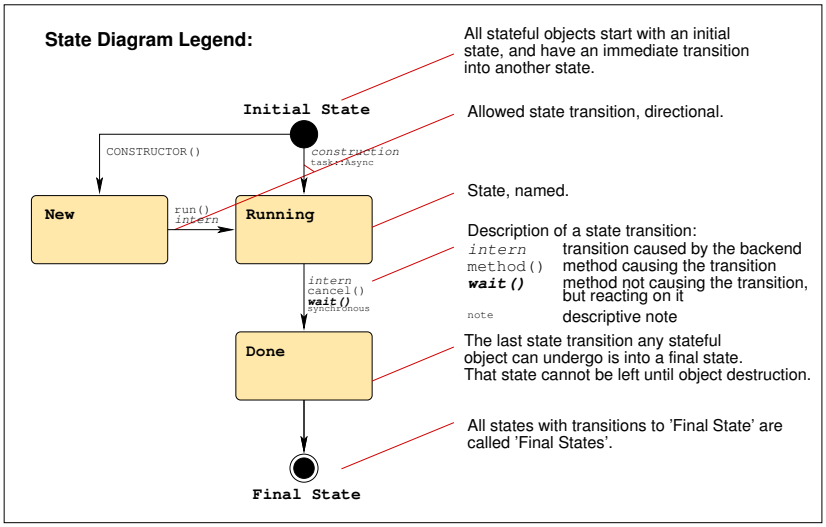

Figure 1: The SAGA state diagrams follow the notations shown here.

2.8 Execution Semantics and Consistency Model

A topic related to concurrency control concerns execution semantics of the operations invoked via SAGA's API calls. Unlike Section 2.6, here we are dealing with the complete execution "chain," reaching from the client API to the server side, based on whichever service or middleware layer is providing access to the server itself.

SAGA API calls on a single service or server can occur concurrently with (a) other tasks from the same SAGA application, (b) tasks from other SAGA applications, or also (c) calls from other, independently developed (non-SAGA) applications. This means that the user of the SAGA API should not rely on any specific execution order of concurrent API calls. However, implementations MUST guarantee that a synchronous method is indeed finished when the method returns, and that an asynchronous method is indeed finished when the task instance representing this method is in *Finished or Done a final* state. Further control of execution order, if needed, has to be enforced via separate concurrency control mechanisms, preferably provided by the services themselves, or on application level.

Most SAGA calls will invoke services that are remote to the application program, hence becoming vulnerable to errors caused by remote (network-based) invocation. Therefore, implementors SHOULD strive to implement "At Most Once" semantics, enforcing that, in case of failures, an API call either fails (does not get executed), or succeeds, but never gets executed more than once.

This seems to be (a) generally supported by most grid middleware, (b) implementable in distributed systems with reasonable effort, and (c) useful and intuitively expected by most end users. Any deviation from these semantics MUST be carefully documented by the implementation.

Beyond this, the SAGA API specification does *not* prescribe any consistency model for its operations, as we feel that this would be very hard to implement across different middleware platforms. A SAGA implementation MAY specify some consistency model, which MUST be documented. A SAGA implementation SHOULD always allow for application level consistency enforcement, for example by use of of application level locks and mutexes.

2.9 Optimizing Implementations, Latency Hiding

Distributed applications are usually very sensistive to communication latencies. Several use cases in SAGA explicitly address this topic, and require the SAGA API to support (a) asynchronous operations, and (b) bulk operations, as both are commonly accepted latency hiding techniques. The SAGA task model (see Section 3.10) provides asynchronous operations for the SAGA API. Bulk operations have no explicit expression in SAGA. Instead, we think that implementations should be able to exploit the concurrency information available in the SAGA task model to transparently support bulk optimizations. In particular, the `saga::task_container` allows to run multiple asynchronous operations at the same time – implementations are encouraged to apply bulk optimizations in that situation. A proof-of-concept implementation in C++ demonstrates that bulk optimizations for task containers are indeed implementable, and perform very well [13]. We feel that this leaves the SAGA API simple, and at the same time allows for performance critical use cases.

Other optimizations are more explicit in the API, most notably the additional I/O operations for the `saga::file` class – those are described in more detail in Section 4.3.

Implementations are encouraged to exploit further optimizations; these MUST NOT change the semantics of the SAGA API though.

2.10 Configuration Management

The SAGA CORE WG spent a significant amount of discussion on deployment and configuration issues, and could not, as of yet, come to a complete agreement on these. More specifically we see the following problems related to the use of SAGA API implementations: Defining deployment and configuration related parts of an API normatively raises a number of issues, such as:

- As different SAGA **implementations** bind to different middleware, that middleware might need configuration information, such as the location of a GridRPC config file (see [19]), or the location of a service endpoint.
- If such configuration information **is** to be provided by the end user, the end user might face, eventually, a plethora of SAGA implementation *and middleware* specific configuration files, or environment variables, or other configuration mechanisms, which *would* break the SAGA abstraction from the middleware for the end user.
- Defining a SAGA configuration file format might succeed syntactically (e.g. ini file format), but must fail semantically, as it will be impossible to foresee on which middleware SAGA gets implemented, and to know which configuration information that middleware requires.

This leaves the dilemma that a configuration mechanism seems impossible to define generically, but by leaving it undefined, we break the abstraction SAGA is supposed to provide to the end user.

For the time being, *we leave* this problem *is left* to (a) the middleware developers, (b) to the SAGA implementors, and (c) to the SAGA deployment (i.e. system administrators). *We hope that* Experience gathered by these groups will *hopefully* allow *us* to revise this topic, and to define a generic, simple, *and* abstract approach to the configuration problem.

2.11 The 'URL Problem'

The end user might expect the SAGA API, as a high level and simple API, to handle protocol specific issues transparently. In particular, she might expect that SAGA gracefully and intelligently handles a URL such as

```
http://host.net//tmp/file
```

even if HTTP as *a* protocol is, in fact, not available at `host.net`, but for example the FTP protocol is.

However, this innocently looking problem has far reaching consequences, and in fact is, to the best of our knowledge, unresolved. Consider the following server setup on host.net:

FTP server root:	`/var/ftp/pub/`
HTTP server root:	`/var/http/htdocs/`

The entities described by the two URLs

```
http://host.net//tmp/file
ftp://host.net//tmp/file
```

hence refer to different files on host.net! Even worse: it might be (and often is) impossible to access the HTTP file space via the FTP service, and vice versa.

Similar considerations hold *for absolute file names, and* for file names relative to the user's home directory. Consider:

```
http://host.net/~user/tmp/file
```

This URL may point to

```
file:////home/user/public_html/tmp/file
```

and not, as could have been expected, to

```
file:////home/user/tmp/file
```

Hence, a reliable translation of URLs between different protocols (*or protocol* schemes) is only possible, if the exact server setup of all affected protocol serving services is known. This knowledge is often not available.

Further, even if a correct translation of protocols and hence URLs suceeds, there is no guarantee that the referred file is actually available via this protocol, with the same permissions etc. – this again depends on the service configuration.

SAGA 'solution' to the 'URL Problem'

1. A SAGA compliant implementation MAY be able to transparently translate URLs, but is not required to do so. Further, this behaviour CAN vary during the runtime of the program.
2. *A SAGA compliant implementation MUST provide the* ***`translate`*** *method as part of the* ***`saga::url`*** *class. That method allows the end user to check if a specific URL translation can be performed.*
3. The SAGA API specification allows the use of the placeholder 'any' (as in `any://host.net/tmp/file`). A SAGA compliant implementation MAY be able to choose a suitable protocol automatically, but CAN decline the URL with an `IncorrectURL` exception.
4. Abstract name spaces, such as the name space used by replica systems, or by grid file systems, hide this problem efficiently and transparently from the end user. We encourage implementations to use such name spaces.
5. A URL which cannot be handled for the stated reasons MUST cause the exception `IncorrectURL` to be thrown. Note that this holds only for

those cases where a given URL cannot be handled *as such*, e.g. because the protocol is unsupported, `any://` cannot be handled, or a necessary URL translation failed. The detailed error message SHOULD give advice to the end user which protocols are supported, and which types of URL translations can or cannot be expected to work. *The `IncorrectURL` exception is thus listed on all methods which handle URLs as parameters, but is not individually motivated in the detailed method specifications.*

6. Any other error related to the URL (e.g. *file at service is not available invalid file name*) MUST be indicated by the exceptions as listed in the method specifications in this document *(in most cases a `BadParameter` exception) is applicable*.

We are aware that this 'solution' is sub-optimal, but we also think that, if cleverly implemented with the help of information services, service level setup information, and global name spaces, this approach can simplify the use of the SAGA API significantly. We will carefully watch the work of related OGF groups, such as the global naming efforts in the Grid FileSystem Working Group (GFS-WG), and will revise this specification if any standard proposal is put forward to address the described problem.

Note that SAGA, unlike other Grid APIs such as the GAT[2], is fully adopting RFC 3986[5]: URLs which include a scheme can, according to that RFC, not express relative locations. The following two URLs are thus expected to point to the same location:

```
gridftp://remote.host.net/bin/date
gridftp://remote.host.net//bin/date
```

2.12 Miscellaneous Issues

2.12.1 File Open Flags

For files, flags are used to specify if an `open` is truncating, creating, and/or appending to an existing entity. For jobs, and in particular for file staging, the LSF scheme is used (e.g. `'url >> local_file'` for appending a remote file to a local one after staging). We are aware of this seeming inconsistency. However, we think that a forceful unification of both schemes would be more awkward to use, and at the same time less useful.

2.12.2 Byte Ordering

Applications on grids as inherent homogeneous environments will often face different native byte orders on different resources. In general, SAGA always operates in the locally native byte ordering scheme, unless explicitely notified. The byte oriented I/O interfaces (files and streams) are naturally ignorant to the byte ordering. Finally, any byte order conversion on data exchange beween two SAGA applications, e.g. by using files, streams or remote procedure calls, must be taken care of in application space, unless noted otherwise.

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 2 on Page 35. This figure also marks which interfaces are *dominating* *part of* the SAGA **Look-&-Feel**, and which classes are combined into packages.

This and the next section form the normative part of the SAGA Core API specification. It has one subsection for each package, starting with those interfaces that define the SAGA **Look-&-Feel** *(top level interfaces first)*, followed by the various, **capability-providing** packages: job management, name space management, file management, replica management, streams, and remote procedure call.

The SAGA **Look-&-Feel** is defined by a number of classes and interfaces which ensure the non-functional properties of the SAGA API (see [18] for a complete list of non-functional requirements). These interfaces and classes are intended to be used by the functional SAGA API packages, and are hence thought to be orthogonal to the functional scope of the SAGA API.

SAGA implementations should be able to implement the SAGA **Look-&-Feel** API packages independent of the grid middleware backend. *That is* *This*, however, *not guaranteed to be the case.* *might not always be possible, at least to a full extent.* In particuar Monitoring and Steering, but also *Attributes and* asynchronous operations, may need explicit support from the backend system. As such, methods in these four packages MUST be expected to throw a `NotImplemented` exception, in accordance with the SAGA implementation compliance **guidelines** given in *Section 2.4*. *The `NotImplemented` exception is listed in the respective method description details, but, unlike other listed exceptions, not separately motivated.* Similarly, the `IncorrectURL` exception is listed when appropriate, but is not, in general, separately motivated or detailed – the semantic conventions for this exception are as defined in Section 2.11.

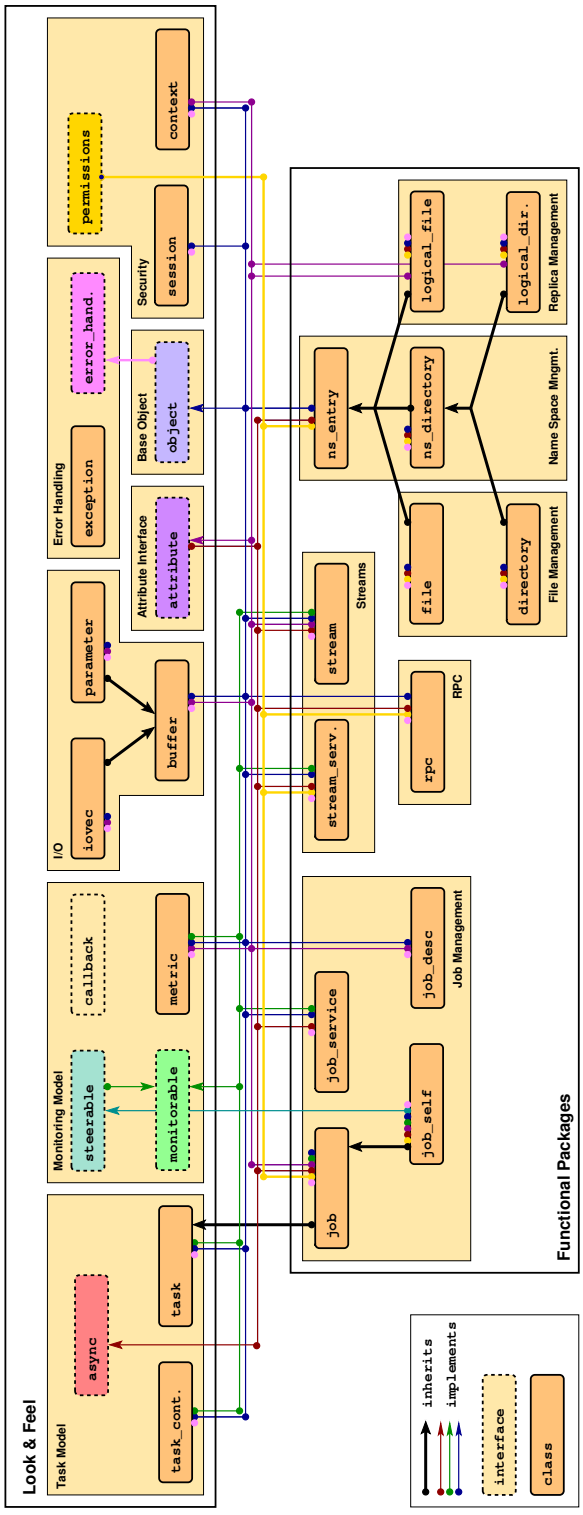

Figure 2: The SAGA class and interface hierarchy.

3.1 SAGA Error Handling

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

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 facilities* *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.* Bindings for languages without exception handling capabilities MUST stick to the `error_handler` interface described here, but MAY define additional language-native means for error reporting.

For asynchronous operations, the error handler interface is provided by the task instance performing the operation, and not by the object which created the task.

If an error occurs during object creation, then the error handler interface of the session the object was to be created in will report the error.

For objects implementing the `error_handler` interface, each method invocation on that object resets any error caused by a previous method invocation on that object.

In languages bindings where this is appropriate, some API methods return POSIX `errno` codes for errors. This is *true* *the case* in particular for `read()`, `write()` and `seek()`, for `saga::file` and `saga::stream`. The respective method descriptions provide explicit details of how `errno` error codes are utilized. In any case, *whenever numerical* *the user* `errno` codes are *to be used as they are used in* *used, they have to be conforming to* POSIX.1 [21].

Any other details of the error handling mechanisms will be defined in the respective language bindings, if required.

3.1.1 Specification

```
  package saga.error
  {
    class exception
    {
!     CONSTRUCTOR          (in  object         obj,
                            in  string         message,
+                           out exception      e);
+     CONSTRUCTOR          (in  string         message,
+                           out exception      e);
      DESTRUCTOR           (void);

-     what                 (out string         message);
      get_message          (out string         message);
!     get_object           (out object         obj);
    }


+   class NotImplemented       : extends saga::exception { }
+   class IncorrectURL         : extends saga::exception { }
+   class BadParameter         : extends saga::exception { }
+   class AlreadyExists        : extends saga::exception { }
+   class DoesNotExist         : extends saga::exception { }
+   class IncorrectState       : extends saga::exception { }
+   class PermissionDenied     : extends saga::exception { }
+   class AuthorizationFailed  : extends saga::exception { }
+   class AuthenticationFailed : extends saga::exception { }
+   class Timeout              : extends saga::exception { }
+   class NoSuccess            : extends saga::exception { }


    interface error_handler
    {
      has_error            (out boolean        has_error);
      get_error            (out exception      error);
    }
  }
```

3.1.2 Specification Details

SAGA provides a set of well-defined *exceptions (error states)* which MUST be supported by the implementation. As to wether these error states are critical, non-critical or fatal depends on, (a) the specific implementation (one implementation might be able to recover from an error while another implementation might not), and (b) the specific application use case (e.g. the error 'file does not exist' may or may not be fatal, depending on whether the application really needs information from that file).

Several *In laguage bindings where this is appropriate, some* SAGA methods do not raise exceptions on certain error conditions, but return an error code *instead*. For example, `file.read()` might return an error code indicating that a *not enough* data *is* available right now. The error codes used in SAGA are based on the definitions for `errno` as defined by POSIX, and MUST be used in a semantically identical manner.

This specification defines the set of allowed exceptions for each method explicitly – that *this* set is normative The SAGA specification defines which exceptions can be thrown by which method. *: other SAGA exceptions MUST NOT be thrown on these methods. Also, implementations MUST NOT specify or use other SAGA exceptions than listed in this specification.*

For example, a certain implementation might have authorization on session creation, and could throw an `AuthorizationFailed` *exception on attempts to create a session – even though this is not specified in the SAGA specification. New SAGA exception types, however, SHOULD NOT be defined by the implementation.*

Additionally, an implementation MAY throw other, non-SAGA exceptions, e.g. on system errors, resource shortage etc. SAGA implementations MUST, however, translate grid middleware-specific exceptions and error conditions into SAGA exceptions whenever possible, in order to avoid middleware specific exception handling on applications level – that would clearly contradict the intent of SAGA to be middleware independent.

In the SAGA language bindings, exceptions are either derived from the base SAGA exception types, or are error codes with that specific name etc. The exact rendering is language binding specific; for details, see the language bindings. *Note that the detailed description for* `saga::exception` *below does not list the* `CONSTRUCTOR`*s and* `DESTRUCTOR`*s for all exception classes individually, but only for the base exception class. The individual exception classes MUST NOT add syntax or semantics to the base exception class.*

The string returned by `what()` and `get_message()` MUST be formatted as follows:

```
"<ExceptionName>: message"
```

where `<ExceptionName>` MUST match the literal exception names as defined in this document, and `message` SHOULD be a detailed, human readable description of the cause of the exception.

For implementations with multiple middleware bindings, it can be difficult to provide detailed and conclusive error messaging with a single exception. To support such implementations, language bindings MAY allow nested exceptions. The outermost exception MUST, however, follow the syntax and semantics guidelines described above. Implementations of such bindings which only bind to a single backend MUST support the defined interface for nested exceptions as well, in order to keep the application independent of the specifics of the SAGA implementation, but will then in general not be able to return lower-level exceptions.

The exceptions available in SAGA are listed below, with a number of explicit examples on when exceptions should be thrown. These examples are not normative, but merely illustrative. This list of exceptions is sorted, with the most specific exceptions *are* listed first and least specific last. The most specific exception possible (i.e. applicable) MUST be thrown on all error conditions. *This means that if multiple exceptions are applicable to an error condition (e.g.* ***`PermissionDenied`*** *and* ***`NoSuccess`*** *for opening a file with incorrect permissions), then that exception MUST be thrown which gives more specific information about the respective error condition: e.g.,* ***`PermissionDenied`*** *describes the error condition much more explicitly than a generic* ***`NoSuccess`****.*

- `NotImplemented`

If a method is specified in the SAGA API, but cannot be provided by a specific SAGA implementation, this exception MUST be thrown. *Object constructors can also throw that exception, if the respective object is not implemented by that SAGA implementation at all.* See also the notes about compliant implementations in *Section 2.4*.

Examples:

- An implementation based on Unicore might not be able to provide streams. The `saga::stream_server` constructor should throw a NotImplemented exception for *that such an* implementation.

The `IncorrectSession` exception was removed due to some implications of the new task model.

- `IncorrectURL`

This exception is thrown if a method is invoked with **a** URL argument that could not be handled. This error specifically indicates that an implementation **cannot**

handle the specified protocol, or *that* access to the specified entity via the given protocol is impossible. The exception MUST NOT be used to indicate any other error condition. See also the notes to 'The URL Problem' in *Section 2.11*.

Examples:

- An implementation based on gridftp might be unable to handle http-based URLs sensibly, and might be unable to translate them into *gridftp* based URLs internally. The implementation should then throw an `IncorrectURL` exception if it encounters **a** http-based URL.
- *A URL is well formed, but includes characters or path elements which are not supported by the SAGA implementation or the backend. Then, an `IncorrectURL` exception is thrown, with detailed information on why the URL could not be used.*

• `BadParameter`

This exception indicates that at least one of the parameters of the method call is ill-formed, invalid, out of **bounds** or otherwise not usable. The error message MUST give specific information on what parameter caused the exception, and why.

Examples:

- a specified context type is not supported by the implementation
- a file name specified is invalid, e.g. too long, or contains characters which are not allowed
- an ivec for scattered read/write is invalid, e.g. has offsets which are out of **bounds**, or *refer to* non-allocated buffers
- a buffer to be written and the specified lengths are incompatible
- an enum specified is not known
- flags specified are incompatible (`ReadOnly` *and* `Truncate`)

• `AlreadyExists`

This exception indicates that an operation cannot succeed because an entity to be created or registered already exists or is already registered, and cannot be overwritten. Explicit flags on the method invocation may allow the operation to succeed, e.g. if they indicate that Overwrite is allowed.

Examples:

- a target for a file move already exists

- a file to be created already exists
- a name to be added to a logical file is already known
- a metric to be added to a object has the same name as an existing metric on that object

• `DoesNotExist`

This exception indicates that an operation cannot succeed because a required entity is missing. Explicit flags on the method invocation may allow the operation to succeed, e.g. if they indicate that Create is allowed.

Examples:

- a file to be moved does not exist
- a directory to be listed does not exist
- a name to be deleted is not in a replica set
- a metric asked for is not known to the object
- a context asked for is not known to the session
- a task asked for is not in a task container
- a job asked for is not known by the backend
- **an** attribute asked for is not supported

ReadOnly and WriteOnly exception are now part of the PermissionDenied exception, because of the changes to the SAGA permission model.

• `IncorrectState`

This exception indicates that the object a method was called on is in a state where that method cannot possibly succeed. A change of state might allow the method to succeed with the same set of parameters.

Examples:

- calling read on a stream which is not connected
- calling run on a task which was canceled
- calling resume on a job which is not suspended

• `PermissionDenied`

An operation failed because the **identity** used for the operation did not have

sufficient permissions to perform the operation successfully. The authentication and authorization steps have been completed successfully.

Examples:

- attempt to change or set a ReadOnly attribute
- attempt to change or update a ReadOnly metric
- calling write on a file which is opened for read only
- *calling read on a file which is opened for write only*
- although a user could login to a remote host via GridFTP and could be mapped to a local user, the write on /etc/passwd failed.

Examples:

- `AuthorizationFailed`

An operation failed because none of the available contexts of the used session could be used for successful **authorization**. That error indicates that the resource could not be accessed at all, and not that an operation was not available due to restricted permissions. The authentication step has been completed successfully.

The differences between AuthorizationFailed and PermissionDenied are, admittedly, subtle. Our intention for introducing both exceptions was to allow to distinguish between administrative authorization failures (**on** VO and DN level), and backend related authorization failures (which can often be resolved on user level).

The AuthorizationFailed exception SHOULD be thrown when the **backend** does not allow the execution of the requested operation at all, whereas the PermissionDenied exception SHOULD be thrown if the operation was executed, but failed due to insufficient privileges.

Examples:

- although a certificate was valid on a remote GridFTP server, the distinguished name could not be mapped to a valid local user id. A call to file.copy() should then throw an AuthorizationFailed exception.

- `AuthenticationFailed`

An operation failed because none of the available session contexts could successfully be used for authentication.

Examples:

- a remote host does not accept a X509 certificate because the respective CA is unknown there. A call to file.copy() should then throw an AuthenticationFailed exception.

• `Timeout`

This exception indicates that a remote operation did not complete successfully because the network communication or the remote service timed out. The time **waited before an implementation** raises a `Timeout` exception depends on implementation and backend details, and SHOULD be documented by the implementation. *That* *This* exception MUST NOT be thrown if a timed `wait()` or similar method times out. *- that is indicated by the method's return value, and does not pose an error condition.* *The latter is not an error condition and gets indicated by the method's return value.*

Examples:

- a remote file authorization request timed out
- a remote file read operation **timed** out
- a host name resolution timed out
- a started file transfer stalled and timed out
- **an asynchronous** file transfer stalled and timed out

• `NoSuccess`

This exception indicates that an operation failed semantically, e.g. the operation was not successfully performed. This exception is the least specific exception defined in SAGA, and CAN be used for all error conditions which do not indicate a more specific exception specified above. *The error message SHOULD always contain some further detail, describing the circumstances which caused the error condition.*

Examples:

- a once open file is not available right now
- a backend response cannot be parsed
- *a remote procedure call failed due to a corrupted parameter stack*
- a file copy was interrupted mid-stream, due to shortage of disk space

Class `exception`

This is the exception base class inherited by all exceptions thrown by a SAGA object implementation.

Note that `saga::exception` does not implement the `saga::object` interface.

```
    - CONSTRUCTOR
      Purpose:  create the exception
!     Format:   CONSTRUCTOR   (in  object         obj,
+                              in  string         message
                               out exception      e);
!     Inputs:   obj:          the object associated with the
                              exception.
+               message:      the message to be associated
+                             with the new exception
```

```
+     InOuts:   -
      Outputs:  e:             the newly created exception
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   -
+     Notes:    -

+
+   - CONSTRUCTOR
+     Purpose:  create the exception,  without associating
+               a saga object instance
+     Format:   CONSTRUCTOR   (in  string         message
+                              out exception      e);
+     Inputs:   message:       the message to be associated
+                              with the new exception
+     InOuts:   -
+     Outputs:  e:             the newly created exception
+     PreCond:  -
+     PostCond: -
+     Perms:    -
+     Throws:   -
+     Notes:    -
+

    - DESTRUCTOR
      Purpose:  destroy the exception
      Format:   DESTRUCTOR    (in  exception e);
!     Inputs:   e:             the exception to destroy
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   -
+     Notes:    -

-
-   - what
-     what is an alias for get_message.
-

    - get_message
!     Purpose:  gets the message associated with the exception
      Format:   get_message   (out string message);
      Inputs:   -
```

```
+     InOuts:   -
!     Outputs:  message:       the error message
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   -
      Notes:    - the returned string MUST be formatted as
!                 described earlier in this section.


    - get_object
      Purpose:  gets the SAGA object associated with exception
!     Format:   get_object     (out object obj);
      Inputs:   -
+     InOuts:   -
!     Outputs:  obj:            the object associated with the
                                exception
+     PreCond:  - an object was associated with the exception
+               during construction.
+     PostCond: -
+     Perms:    -
!     Throws:   DoesNotExist
+               NoSuccess
      Notes:    - the returned object is a shallow copy of the
                  object which was used to call the method which
                  caused the exception.
!                 - if the exception is raised in a task, e.g. on
                  task.rethrow(), the object is the one which the
+                 task was created from.  That allows the
+                 application to handle the error condition
+                 without the need to always keep track of
+                 object/task relationships.
                  - an 'DoesNotExist' exception is thrown when no
                  object is associated with the exception, e.g.
!                 if an 'NotImplemented' exception was raised
                  during the construction of an object.
```

Interface `error_handler`

The ***`error_handler`*** *interface allows the application to retrieve exceptions. An alternative approach would be to return an error code for all method invocations. This, however, would put a significant burdon on languages with exception handling, and would also complicate the management of return values. Language*

bindings for languages with exception support will thus generally not implement the ***error_handler*** *interface, but use exceptions instead.*

Implementations which are using the interface maintain an internal error state for each class instance providing the interface. That error state is **`false`** *by default, and is set to* **`true`** *whenever an method invocation meets an error condition which would, according to this specification, result in an exception to be thrown.*

The error state of an object instance can be tested with **`has_errror()`**, *and the respective exception can be retrieved with* **`get_error()`**. *Any one of these calls clears the error state (i.e. resets it to* **`false`**). *Note that there is no other mechanism to clear an error state – that means in particular that any successful method invocation on the object leaves the error state unchanged. If two or more subsequent operations on an object instance fail, then only the last exception is returned on* **`get_error()`**. *That mechanism allows to execute a number of calls, and to check if they resulted in any error condition, somewhat similar to* **`try/catch`** *statements in languages with exception support. However, it must be noted that an exception does not cause subsequent methods to fail, and does not inhibit their execution.*

If **`get_error()`** *is called on an instance whose error state is* **`false`**, *an* **`IncorrectState`** *exception is returned, which MUST state explicitely that the* **`get_error()`** *method has been invoked on an object instance which did not encounter an error condition.*

```
+   - has_error
+     Purpose:  tests if an object method caused an exception
+     Format:   has_error     (out bool       has_error);
+     Inputs:   -
+     InOuts:   -
+     Outputs:  has_error:     indicates that an exception was
+                              caught.
+     PreCond:  -
+     PostCond: - the internal error state is false.
+     Perms:    -
+     Throws:   -
+     Notes:    -
+
+
+   - get_error
+     Purpose:  retrieve an exception catched during a member
+               method invocation.
+     Format:   get_error     (out exception  e);
+     Inputs:   -
+     InOuts:   -
+     Outputs:  e:             the caught exception
```

```
+     PreCond:  - the internal error state is true.
+     PostCond: - the internal error state is false.
+     Perms:    -
+     Throws:   NotImplemented
+               IncorrectURL
+               BadParameter
+               AlreadyExists
+               DoesNotExist
+               IncorrectState
+               PermissionDenied
+               AuthorizationFailed
+               AuthenticationFailed
+               Timeout
+               NoSuccess
+     Notes:    - the method throws the error/exception it is
+                 reporting about.
+               - an 'IncorrectState' exception is also thrown
+                 if the internal error state is false.
```

3.1.3 Examples

Code Example

```
1  ////////////////////////////////////////////////////////////////
2  //
3  // C++ examples for exception handling in SAGA
4  //
5  ////////////////////////////////////////////////////////////////
6
7  ////////////////////////////////////////////////////////////////
8  //
9  // simple exception handling
10 //
11 int main ()
12 {
13   try
14   {
15     saga::file f ("file://localhost/etc/passwd");
16     f.copy ("file:///usr/tmp/passwd.bak");
17   }
18
19   catch ( const saga::exception::PermissionDenied & e )
20   {
21     std::cerr << "SAGA error: No Permissions!" << std::endl;
22     return (-1);
```

```
23    }
24
25    catch ( const saga::exception & e )
26    {
27      std::cerr << "SAGA error: "
28                << e.get_message ()
29                << std::endl;
30      return (-1);
31    }
32
33    return (0);
34  }
35
36
37  ////////////////////////////////////////////////////////////////
38  //
39  // exception handling for tasks
40  //
41  int main ()
42  {
43
44    saga::file f ("file://localhost/etc/passwd");
45
46    saga::task t = f.copy <saga::task::Async>
47                          ("file:///usr/tmp/passwd.bak");
48
49    t.wait ();
50
51    if ( t.get_state () == saga::task::Failed )
52    {
53      try {
54        task.rethrow ();
55      }
56      catch ( const saga::exception & e )
57      {
58        std::cout << "task failed: "
59                  << e.get_message ()
60                  << std::endl;
61      }
62      return (-1);
63    }
64    return (0);
65  }
```

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 [16].

Note that there are no object IDs for the various SAGA exceptions, but only one ID for the `saga::exception` *base class. Also, it is not possible to inspect a SAGA object instance for the availability of certain SAGA interfaces, as they are fixed and well defined by the SAGA specification. Language bindings MAY, however, add such inspection, if that is natively supported by the language.*

3.2.1 Specification

```
  package saga.object
  {
    enum object_type
    {
-     Unknown          =  -1,
      Exception        =   1,
+     URL              =   2,
+     Buffer           =   3,
      Session          =   4,
      Context          =   5,
      Task             =   6,
      TaskContainer    =   7,
!     Metric           =   8,
!     NSEntry          =   9,
!     NSDirectory      =  10,
+     IOVec            =  11,
!     File             =  12,
!     Directory        =  13,
!     LogicalFile      =  14,
!     LogicalDirectory =  15,
!     JobDescription   =  16,
!     JobService       =  17,
!     Job              =  18,
!     JobSelf          =  19,
```

```
!     StreamService    =  20,
!     Stream           =  21,
+     Parameter        =  22,
+     RPC              =  23,
-     Multiplexer      =  24
    }


    interface object : implements saga::error-handler
    {
      get_id       (out string      id     );
      get_type     (out object_type type   );
!     get_session  (out session     s      );

      // deep copy
      clone        (out object      clone  );
    }
  }
```

3.2.2 Specification Details

Enum `object_type`

The SAGA ***`object_type`*** *enum allows for inspection of SAGA object instances. This, in turn, allows to treat large numbers of SAGA object instances in containers, without the need to create separate container types for each specific SAGA object type. Bindings to languages that natively support inspection on object types MAY omit this enum and the* ***`get_type()`*** *method.*

SAGA extensions which introduce new SAGA objects (i.e. introduce new classes which implement the ***`saga::object`*** *interface) MUST define the appropriate* ***`object_type`*** *enums for inspection. SAGA implementations SHOULD support these enums for all packages which are provided in that implementation, even for classes which are not implented.*

Interface `object`

```
    - get_id
      Purpose:  query the object ID
      Format:   get_id               (out string id);
```

```
      Inputs:   -
+     InOuts:   -
!     Outputs:  id:                   uuid for the object
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   -
+     Notes:    -


    - get_type
      Purpose:  query the object type
      Format:   get_type             (out object_type type);
      Inputs:   -
+     InOuts:   -
!     Outputs:  type:                 type of the object
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   -
+     Notes:    -


    - get_session
      Purpose:  query the objects session
      Format:   get_session          (out session s);
      Inputs:   -
+     InOuts:   -
!     Outputs:  s:                    session of the object
+     PreCond:  - the object was created in a session, either
+               explicitly or implicitly.
+     PostCond: - the returned session is shallow copied.
+     Perms:    -
!     Throws:   DoesNotExist
      Notes:    - if no specific session was attached to the
                  object at creation time, the default SAGA
                  session is returned.
!                 - some objects do not have sessions attached,
                  such as job_description, task, metric, and the
                  session object itself.  For such objects, the
!                 method raises a 'DoesNotExist' exception.


    // deep copy:
    -------------
```

```
    - clone
      Purpose:  deep copy the object
      Format:   clone                (out object clone);
      Inputs:   -
+     InOuts:   -
!     Outputs:  clone:                the deep copied object
+     PreCond:  -
+     PostCond: - apart from session and callbacks, no other
+                 state is shared between the original object
+                 and it's copy.
+     Perms:    -
+     Throws:   NoSuccess
      Notes:    - that method is overloaded by all classes
                  which implement saga::object, and returns
!                 a deep copy of the respective class type
!                 (the method is only listed here).
!               - the method SHOULD NOT cause any backend
                  activity, but is supposed to clone the client
                  side state only.
!               - for deep copy semantics, see Section 2.
```

3.2.3 Examples

Code Example

```
1   // c++ example
2
3   // have 2 objects, streams and files, and do:
4   //  - read 100 bytes
5   //  - skip 100 bytes
6   //  - read 100 bytes
7
8   int  out;
9   char data1[100];
10  char data2[100];
11  char data[100];
12
13  saga::buffer buf1 (data1, 100);
14  saga::buffer buf2 (data2, 100);
15  saga::buffer buf;
16
17  // create objects
18  saga::file   f (url[1]);
19  saga::stream s (url[2]);
20
21  // f is opened at creation, s needs to be connected
```

```
22  s.connect ();
23
24  // create tasks for reading first 100 bytes ...
25  saga::task t1 = f.read <saga::task> (100, buf1);
26  saga::task t2 = s.read <saga::task> (100, buf2);
27
28  // create and fill the task container ...
29  saga::task_container tc;
30
31  tc.add (t1);
32  tc.add (t2);
33
34  // ... and wait who gets done first
35  while ( saga::task t = tc.wait (saga::task::Any) )
36  {
37     // depending on type, skip 100 bytes then create a
38     // new task for the next read, and re-add to the tc
39
40     switch ( t.get_object().get_type () )
41     {
42       case saga::object::File :
43         // point buf to results
44         buf = buf1;
45
46         // get back file object
47         saga::file f = saga::file (t.get_object ());
48
49         // skip for file type (sync seek)
50         saga::file (f.seek (100, SEEK_SET);
51
52         // create a new read task
53         saga::task t2 = f.read <saga::task> (100, buf1));
54
55         // add the task to the container again
56         tc.add (t2);
57
58         break;
59
60       case saga::object::Stream :
61         // point buf to results
62         buf = buf2;
63
64         // get back stream object
65         saga::stream s = saga::stream (t.get_object ());
66
67         // skip for stream type (sync read and ignore)
68         saga::stream (s.read (100, buf2);
69
70         // create a new read task
71         saga::task t2 = s.read <saga::task> (100, buf2));
```

```
72
73          // add the task to the container again
74          tc.add (t2);
75
76          break;
77
78        default:
79          throw exception ("Something is terribly wrong!");
80      }
81
82      std::cout << "found: '" << out << " bytes: "
83                              << buf.get_data ()
84                              << std::endl;
85
86      // tc is filled again, we run forever, read/seeking from
87      // whatever we find after the wait.
88    }
```

3.3 SAGA URL Class

This whole section has been added to the specification!

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

- simplify the construction and the parsing of URLs on application level,
- allow for sanity checks within and outside the SAGA implementation,
- simplify and unify the signatures of SAGA calls which accept URLs,

a SAGA URL class is used. This class provides means to set and access the various elements of a URL. The class parses the URL in conformance to RFC-3986 [5].

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

```
package saga.url
{
  class url : implements   saga::object
           // from object  saga::error_handler
  {
    CONSTRUCTOR   (in  string     url       ,
                   out buffer     obj       );
    DESTRUCTOR    (in  buffer     obj       );

    set_string    (in  string     url       = "");
    get_string    (out string     url       );

    set_scheme    (in  string     scheme    = "");
    get_scheme    (out string     scheme    );

    set_host      (in  string     host      = "");
    get_host      (out string     host      );

    set_port      (in  int        port      = "");
    get_port      (out int        port      );
```

```
      set_fragment  (in  string     fragment  = "");
      get_fragment  (out string     fragment  );

      set_path      (in  string     path      = "");
      get_path      (out string     path      );

      set_query     (in  string     query     = "");
      get_query     (out string     query     );

      set_userinfo  (in  string     userinfo  = "");
      get_userinfo  (out string     userinfo  );

      translate     (in  string     scheme    ,
                     out url        url       );
    }
  }
```

3.3.1 Specification Details

Class `url`

```
    - CONSTRUCTOR
      Purpose:  create a url instance
      Format:   CONSTRUCTOR          (in  string url = "",
                                      out url    obj);
      Inputs:   url:                  initial URL to be used
      InOuts:   -
      Outputs:  url:                  the newly created url
      PreCond:  -
      PostCond: -
      Perms:    -
      Throws:   NotImplemented
                BadParameter
+               NoSuccess
      Notes:    - if the implementation cannot parse the given
                  url, a 'BadParameter' exception is thrown.
                - this constructor will never throw an
                  'IncorrectURL' exception, as the
                  interpretation of the URL is not part of this
                  class'es functionality.
                - the implementation MAY change the given
```

```
              URL as long as that does not change the
              resource the URL is pointing to.  For
              example, an implementation may normalize the
              path element of the URL.


  - DESTRUCTOR
    Purpose:  destroy a url
    Format:   DESTRUCTOR           (in  url obj);
    Inputs:   obj:                  the url to destroy
    InOuts:   -
    Outputs:  -
    PreCond:  -
    PostCond: -
    Perms:    -
    Throws:   -
    Notes:    -


  - set_string
    Purpose:  set a new url
    Format:   set_string           (in  string url = "");
    Inputs:   url:                  new url
    InOuts:   -
    Outputs:  -
    PreCond:  -
    PostCond: -
    Perms:    -
    Throws:   NotImplemented
              BadParameter
    Notes:    - the method is semantically equivalent to
                destroying the url, and re-creating it with
                the given parameter.
              - the notes for the DESTRUCTOR and the
                CONSTRUCTOR apply.


  - get_string
    Purpose:  retrieve the url as string
    Format:   get_string           (out string url);
    Inputs:   -
    InOuts:   -
    Outputs:  url:                  string representing the url
    PreCond:  -
    PostCond: -
    Perms:    -
```

```
  Throws:   NotImplemented
  Notes:    - the URL may be empty, e.g. after creating the
              instance with an empty url parameter.


- set_*
  Purpose:  set an url element
  Format:   set_<element>       (in  string <element> = "");
            set_scheme          (in  string scheme    = "");
            set_host            (in  string host      = "");
            set_port            (in  int    port      = "");
            set_fragment        (in  string fragment  = "");
            set_path            (in  string path      = "");
            set_query           (in  string query     = "");
            set_userinfo        (in  string userinfo  = "");
  Inputs:   <element>:           new url <element>
  InOuts:   -
  Outputs:  -
  PreCond:  -
  PostCond: - the <element> part of the URL is updated.
  Perms:    -
  Throws:   NotImplemented
            BadParameter
  Notes:    - these calls allow to update the various
              elements of the url.
            - the given <element> is parsed, and if it is
              either not well formed (see RFC-3986), or the
              implementation cannot handle it, a
              'BadParameter' exception is thrown.
            - if the given <element> is empty, it is removed
              from the URL.  If that results in an invalid
              URL, a 'BadParameter' exception is thrown.
            - the implementation MAY change the given
              elements as long as that does not change the
              resource the URL is pointing to.  For
              example, an implementation may normalize the
              path element.


- get_*
  Purpose:  get an url element
  Format:   get_<element>       (out string <element>);
            get_scheme          (out string scheme   );
            get_host            (out string host     );
            get_port            (out int    port     );
            get_fragment        (out string fragment );
```

```
            get_path            (out string path     );
            get_query           (out string query    );
            get_userinfo        (out string userinfo );
  Inputs:   -
  InOuts:   -
  Outputs:  <element>:           the url <element>
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   NotImplemented
  Notes:    - these calls allow to retrieve the various
              elements of the url.
            - the returned <element> is either empty, or
              guaranteed to be well formed (see RFC-3986).


- translate
  Purpose:  translate an URL to a new scheme
  Format:   translate            (in  string scheme,
                                  out url    url);
  Inputs:   scheme:               the new scheme to
                                  translate into
  InOuts:   -
  Outputs:  url:                  string representation of
                                  the translated url
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   NotImplemented
            BadParameter
            NoSuccess
  Notes:    - the notes from section 'The URL Problem' apply.
            - if the scheme is not supported,  a
              'BadParameter' exception is thrown.
            - if the scheme is supported, but the url
              cannot be translated to the scheme, a
              'NoSuccess' exception is thrown.
            - if the url can be translated, but cannot be
              handled with the new scheme anymore, no
              exception is thrown.  That can only be
              detected if the returned string is again used
              in a URL contructor, or with set_string().
            - the call does not change the URL represented
              by the class instance itself, but the
              translation is only reflected by the returned
              url string.
```

3.3.2 Examples

Code Example

```
// C++ URL examples

int main (int argc, char ** argv)
{
  if ( argc < 1 )
    return -1;

  std::string url_string = argv[1];

  try
  {
    saga::url url (url_string);

    cout << "url      : " << url.get_string    () << endl;
    cout << "==================================" << endl;
    cout << "scheme   : " << url.get_scheme    () << endl;
    cout << "host     : " << url.get_host      () << endl;
    cout << "port     : " << url.get_port      () << endl;
    cout << "fragment : " << url.get_fragment  () << endl;
    cout << "path     : " << url.get_path      () << endl;
    cout << "query    : " << url.get_query     () << endl;
    cout << "userinfo : " << url.get_userinfo  () << endl;
    cout << "==================================" << endl;

    url.set_scheme    ("ftp");
    url.set_host      ("ftp.remote.net");
    url.set_port      (1234);
    url.set_fragment  ("");
    url.set_path      ("/tmp/data");
    url.set_query     ("");
    url.set_userinfo  ("ftp:anon");

    cout << "==================================" << endl;
    cout << "scheme   : " << url.get_scheme    () << endl;
    cout << "host     : " << url.get_host      () << endl;
    cout << "port     : " << url.get_port      () << endl;
    cout << "fragment : " << url.get_fragment  () << endl;
    cout << "path     : " << url.get_path      () << endl;
    cout << "query    : " << url.get_query     () << endl;
    cout << "userinfo : " << url.get_userinfo  () << endl;
    cout << "==================================" << endl;
    cout << "url      : " << url.get_string    () << endl;
  }
```

```
44  }
```

3.4 SAGA I/O Buffer

This whole section has been added to the specification!

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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

When calling `set_data()` for application-managed buffers, the implementa-

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

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

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

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

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

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

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

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

3.4.1 Specification

```
package saga.buffer
{
  class buffer : implements   saga::object
              // from object  saga::error_handler
  {
    CONSTRUCTOR (in  array<byte>  data,
                 in  int          size,
                 out buffer       obj);
    CONSTRUCTOR (in  int          size = -1,
                 out buffer       obj);
    DESTRUCTOR  (in  buffer       obj);

    set_size    (in  int          size = -1)
    get_size    (out int          size);

    set_data    (in  array<byte>  data,
                 in  int          size);
    get_data    (out array<byte>  data);

    close       (in  float        timeout = -0.0);
  }
}
```

3.4.2 Specification Details

Class `buffer`

```
  - CONSTRUCTOR
    Purpose:  create an I/O buffer
    Format:   CONSTRUCTOR          (in  array<byte> data,
                                    in  int         size,
                                    out buffer      obj);
    Inputs:   data:                 data to be used
              size:                 size of data to be used
    InOuts:   -
    Outputs:  buffer:               the newly created buffer
    PreCond:  - size >= 0
    PostCond: - the buffer memory is managed by the
```

```
                application.
      Perms:    -
      Throws:   NotImplemented
                BadParameter
+               NoSuccess
      Notes:    - see notes about memory management.
                - if the implementation cannot handle the
                  given data pointer or the given size, a
                  'BadParameter' exception is thrown.
                - later method descriptions refer to this
                  CONSTRUCTOR as 'first CONSTRUCTOR'.


    - CONSTRUCTOR
      Purpose:  create an I/O buffer
      Format:   CONSTRUCTOR          (in  int         size = -1,
                                      out buffer      obj);
      Inputs:   size:                 size of data buffer
      InOuts:   -
      Outputs:  buffer:               the newly created buffer
      PreCond:  -
      PostCond: - the buffer memory is managed by the
                  implementation.
                - if size > 0, the buffer memory is allocated by
                  the implmentation.
      Perms:    -
      Throws:   NotImplemented
                BadParameter
+               NoSuccess
      Notes:    - see notes about memory management.
                - if the implementation cannot handle the
                  given size, a 'BadParameter' exception is
                  thrown.
                - later method descriptions refer to this
                  CONSTRUCTOR as 'second CONSTRUCTOR'.


    - DESTRUCTOR
      Purpose:  destroy a buffer
      Format:   DESTRUCTOR           (in  buffer obj);
      Inputs:   obj:                  the buffer to destroy
      InOuts:   -
      Outputs:  -
      PreCond:  -
      PostCond: -
      Perms:    -
```

```
!     Throws:   -
      Notes:    - if the instance was not closed before, the
!                 DESTRUCTOR performs a close() on the instance,
                  and all notes to close() apply.


    - set_data
      Purpose:  set new buffer data
      Format:   set_data             (in  array<byte> data,
                                      in  int         size);
      Inputs:   data:                 data to be used in buffer
                size:                 size of given data
      InOuts:   -
      Outputs:  -
      PreCond:  -
      PostCond: - the buffer memory is managed by the
                  application.
      Perms:    -
      Throws:   NotImplemented
                BadParameter
                IncorrectState
      Notes:    - the method is semantically equivalent to
                  destroying the buffer, and re-creating it with
                  the first CONSTRUCTOR with the given size.
                - the notes for the DESTRUCTOR and the first
                  CONSTRUCTOR apply.


    - get_data
      Purpose:  retrieve the buffer data
      Format:   get_data             (out array<byte> data);
      Inputs:   -
      InOuts:   -
      Outputs:  data:                 buffer data to retrieve
      PreCond:  -
      PostCond: -
      Perms:    -
      Throws:   NotImplemented
                DoesNotExist
                IncorrectState
      Notes:    - see notes about memory management
                - if the buffer was created as implementation
                  managed (size = -1), but no I/O operation has
                  yet been successfully performed on the buffer,
                  a 'DoesNotExist' exception is thrown.
```

```
- set_size
  Purpose:  set size of buffer
  Format:   set_size           (in  int   size = -1);
  Inputs:   size:               value for size
  InOuts:   -
  Outputs:  -
  PreCond:  -
  PostCond: - the buffer memory is managed by the
              implementation.
  Perms:    -
  Throws:   NotImplemented
            BadParameter
            IncorrectState
  Notes:    - the method is semantically equivalent to
              destroying the buffer, and re-creating it with
              the second CONSTRUCTOR using the given size.
            - the notes for the DESTRUCTOR and the second
              CONSTRUCTOR apply.


- get_size
  Purpose:  retrieve the current value for size
  Format:   get_size           (out int   size);
  Inputs:   -
  InOuts:   -
  Outputs:  size               value of size
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   NotImplemented
            IncorrectState
  Notes:    - if the buffer was created with negative size
              with the second CONSTRUCTOR, or the size was
              set to a negative value with set_size(), this
              method returns '-1' if the buffer was not yet
              used for an I/O operation.
            - if the buffer was used for a successfull I/O
              operation where data have been read into the
              buffer, the call returns the size of the
              memory which has been allocated by the
              implementation during that read operation.


- close
  Purpose:  closes the object
```

```
      Format:   close              (in  float timeout = 0.0);
      Inputs:   timeout             seconds to wait
      InOuts:   -
      Outputs:  -
      Perms:    -
-     PreCond:  - the object was not closed before
+     PreCond:  -
      PostCond: - any operation on the object other than
!                 close() or the DESTRUCTOR will cause
!                 an 'IncorrectState' exception.
      Throws:   NotImplemented
-               IncorrectState
-     Notes:    - it is assumed that a session which opened the
-                 instance can also close it - otherwise the
-                 backend entity must have changed its state,
-                 which causes an 'IncorrectState' exception.
+     Notes:    - any subsequent method call on the object
+                 MUST raise an 'IncorrectState' exception
+                 (apart from DESTRUCTOR and close()).
                - if the current data memory is managed by the
                  implementation, it is freed.
                - close() can be called multiple times, with no
                  side effects.
                - if the current data memory is managed by the
                  application, it is not accessed anymore by the
                  implementation after this method returns.
+               - if close() is implicitly called in the
+                 DESTRUCTOR, it will never throw an exception.
                - for resource deallocation semantics, see
                  Section 2.
                - for timeout semantics, see Section 2.
```

3.4.3 Examples

Code Example

```
1  ////////////////////////////////////////////////////////////////
2  // C++ I/O buffer examples
3  ////////////////////////////////////////////////////////////////
4
5  ////////////////////////////////////////////////////////////////
6  //
7  // general examples
8  //
9  // all following examples ignore the ssize_t return value, which
```

```
10  // should be the number of bytes successfully read
11  //
12  ////////////////////////////////////////////////////////////////
13  {
14    char data[x][y][z];
15    char* target = data + 200;
16    buffer b;
17
18    // the following four block do exactly the same, reading
19    // 100 byte (the read parameter supersedes the buffer size)
20
21    // apps managed memory
22    {
23      b.set_data (target);
24      stream.read (b, 100);
25      printf ("%100s", target);
26    }
27
28    {
29      b.set_data (target, 100);
30      stream.read (b);
31      printf ("%100s", target);
32    }
33
34    {
35      b.set_data (target, 100);
36      stream.read (b, 100);
37      printf ("%100s", target);
38    }
39
40    {
41      b.set_data (target, 200);
42      stream.read (b, 100);
43      printf ("%100s", target);
44    }
45
46
47    // now for impl managed memory
48    {
49      b.set_size (100);
50      stream.read (b);
51      printf ("%100s", b.get_data ());
52    }
53
54    {
55      b.set_size (-1);
56      stream.read (b, 100);
57      printf ("%100s", b.get_data ());
58    }
59
```

```
60      {
61        b.set_size (200);
62        stream.read (b, 100);
63        printf ("%100s", b.get_data ());
64      }
65
66
67      // these two MUST throw, even if there is
68      // enough memory available
69
70      // app managed memory
71      {
72        b.set_data (target, 100);
73        stream.read (b, 200);
74      }
75
76      // impl. managed memory
77      {
78        b.set_size (100);
79        stream.read (b, 200);
80      }
81    }
82
83
84    ////////////////////////////////////////////////////////////////
85    //
86    // the next 4 examples perform two reads from a stream,
87    // first 100 bytes, then 200 bytes.
88    //
89    ////////////////////////////////////////////////////////////////
90
91    // impl managed memory
92    {
93      {
94        buffer b;
95
96        stream.read (b, 100);
97        printf ("%100s", b.get_data ());
98
99        stream.read (b, 200);
100       printf ("%200s", b.get_data ());
101
102     } // b dies here, data are gone after that
103   }
104
105
106   // same as above, but with explicit c'tor
107   {
108     {
109       buffer b (100);
```

```
110       stream.read (b);
111       printf ("%100s", b.get_data ());
112
113       b.set_size (200);
114       stream.read (b);
115       printf ("%200s", b.get_data ());
116
117     } // b dies here, data are gone after that
118   }
119
120
121   // apps managed memory
122   {
123     char   data[x][y][z]; // the complete data set
124     char * target = data; // target memory address to read into...
125     target += offset;     // ... is somewhere in the data space.
126
127     stream.read (buffer (target,       100));
128     stream.read (buffer (target + 100, 200));
129
130     printf ("%300s", target);
131
132     // data must be larger than offset + 300, otherwise bang!
133   }
134
135
136   // same as above with explicit buffer c'tor
137   {
138     char   data[x][y][z]; // the complete data set
139     char * target = data; // target memory address to read into...
140     target += 200;          // ... is somewhere in the data space.
141
142     {
143       buffer b (target, 100);
144       stream.read (b);
145
146       b.set_data (target + 100, 200);
147       stream.read (b);
148
149     } // b dies here.  data are intact after that
150
151     printf ("%300s", target);
152
153     // data must be larger than offset + 300, otherwise bang!
154   }
155
156
157   ////////////////////////////////////////////////////////////////
158   //
159   // the next two examples perform the same reads,
```

```
160  // but switch memory management in between
161  //
162  ////////////////////////////////////////////////////////////////
163
164  // impl managed memory, then apps managed memory
165  {
166    {
167      char [x][y][z] data;
168      char* target = data + 200;
169
170      buffer b;
171
172      // impl managed
173      stream.read (b, 100);
174      printf ("%100s", target);
175
176      b.set_data (target, 200); // impl data are gone after this
177
178      // apps managed
179      stream.read (b);
180      printf ("%200s", target);
181
182    } // b dies here, apps data are ok after that, impl data are gone
183  }
184
185
186  // apps managed memory, then impl managed
187  {
188    {
189      char [x][y][z] data;
190      char* target = data + 200;
191
192      buffer b (target);
193
194      // apps managed
195      stream.read (b, 100);
196      printf ("%100s", target);
197
198      b.set_size (-1);
199
200      // impl managed
201      stream.read (b, 200);
202      printf ("%200s", target);
203
204    } // b dies here, apps data are ok after that, impl data are gone
205  }
206
207
208  ////////////////////////////////////////////////////////////////
209  //
```

```
210   // now similar for write
211   //
212   ////////////////////////////////////////////////////////////////
213
214   ////////////////////////////////////////////////////////////////
215   //
216   // general part
217   //
218   // all examples ignore the ssize_t return value, which should be
219   // the number of bytes successfully written
220   //
221   ////////////////////////////////////////////////////////////////
222   {
223     char data[x][y][z];
224     char* target = data + 200;
225     buffer b;
226
227     // the following four block do exactly the same, writing
228     // 100 byte (the write parameter supersedes the buffer size)
229
230     // apps managed memory
231     {
232       b.set_data (target);
233       stream.write (b, 100);
234     }
235
236     {
237       b.set_data (target, 100);
238       stream.write (b);
239     }
240
241     {
242       b.set_data (target, 100);
243       stream.write (b, 100);
244     }
245
246     {
247       b.set_data (target, 200);
248       stream.write (b, 100);
249     }
250
251
252     // now for impl managed memory
253     {
254       b.set_size (100);
255       memcpy (b.get_data (), target, 100);
256       stream.write (b);
257     }
258
259     {
```

```
260       b.set_size (200);
261       memcpy (b.get_data (), target, 200);
262       stream.write (b, 100);
263     }
264
265
266     // these two MUST throw, even if there is
267     // enough memory available
268
269     // app managed memory
270     {
271       b.set_data (target, 100);
272       stream.write (b, 200); // throws BadParameter
273     }
274
275     // impl. managed memory
276     {
277       b.set_size (100);
278       memcpy (b.get_data (), target, 200); // apps error
279       stream.write (b, 200); // throws BadParameter
280     }
281   }
282
283
284   ////////////////////////////////////////////////////////////////
285   //
286   // the next 4 examples perform two writes to a stream,
287   // first 100 bytes, then 200 bytes.
288   //
289   ////////////////////////////////////////////////////////////////
290
291   // impl managed memory
292   {
293     char   data[x][y][z]; // the complete data set
294     char * target = data; // target memory address to write into...
295     target += offset;     // ... is actually somewhere in the data space.
296
297     {
298       buffer b (200);
299
300       memcpy (b.get_data (), target, 100);
301       stream.write (b, 100);
302
303       memcpy (b.get_data (), target + 100, 200);
304       stream.write (b, 200);
305
306     } // b dies here, data are gone after that
307   }
308
309
```

```
310  // same as above, but using set_size ()
311  {
312    char   data[x][y][z]; // the complete data set
313    char * target = data; // target memory address to write into...
314    target += offset;     // ... is actually somewhere in the data space.
315
316    {
317      buffer b (100);
318      memcpy (b.get_data (), target, 100);
319      stream.write (b);
320
321      b.set_size (200);
322      memcpy (b.get_data (), target + 100, 200);
323      stream.write (b);
324
325    } // b dies here, data are gone after that
326  }
327
328
329  // apps managed memory
330  {
331    char   data[x][y][z]; // the complete data set
332    char * target = data; // target memory address to write into...
333    target += offset;     // ... is actually somewhere in the data space.
334
335    stream.write (buffer (target,       100));
336    stream.write (buffer (target + 100, 200));
337
338    // data must be larger than offset + 300, otherwise bang!
339  }
340
341
342  // same as above with explicit buffer c'tor
343  {
344    char   data[x][y][z]; // the complete data set
345    char * target = data; // target memory address to write into...
346    target += 200;          // ... is actually somewhere in the data space.
347
348    {
349      buffer b (target, 100);
350      stream.write (b);
351
352      b.set_data (target + 100, 200);
353      stream.write (b);
354
355    } // b dies here.  data are intact after that
356
357
358    // data must be larger than offset + 300, otherwise bang!
359  }
```

```
360
361
362   ////////////////////////////////////////////////////////////////
363   //
364   // the next two examples perform the same reads,
365   // but switch memory management in between
366   //
367   ////////////////////////////////////////////////////////////////
368
369   // impl managed memory, then apps managed memory
370   {
371     {
372       char [x][y][z] data;
373       char* target = data + 200;
374
375       buffer b (100);
376
377       // impl managed
378       memcpy (b.get_data (), target, 100);
379       stream.write (b, 100);
380
381       b.set_data (target + 100, 200); // apps managed now
382                                       // impl data are gone after this
383
384       // apps managed
385       stream.write (b);
386
387     } // b dies here, apps data are ok after that, impl data are gone
388   }
389
390
391   // apps managed memory, then impl managed
392   {
393     {
394       char [x][y][z] data;
395       char* target = data + 200;
396
397       buffer b (target);
398
399       // apps managed
400       stream.write (b, 100);
401
402       b.set_size (200); // impl managed now
403       memcpy (b.get_data (), target + 100, 200);
404
405       // impl managed
406       stream.write (b);
407
408     } // b dies here, apps data are ok after that, impl data are gone
```

```
409 }
```

3.5 SAGA Session Management

The session object provides the functionality of a session, *handle,* 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 Specification

```
package saga.session
{
  class session : implements   saga::object
                  // from object  saga::error_handler
  {
    CONSTRUCTOR         (in  bool              default = true,
                         out session           obj);
    DESTRUCTOR          (in  session           obj);

    add_context         (in  context           context);
    remove_context      (in  context           context);
    list_contexts       (out array<context,1> contexts);
  }
}
```

3.5.2 Specification Details

Class `session`

Almost all SAGA objects are created in a SAGA session, and are associated with this (and only this) session for their whole life time.

A session instance to be used on object **instantiation** can **explicitly** be given as first parameter to the SAGA object instantiation call (`CONSTRUCTOR`).

If the session is omitted as first parameter, a default session is used, with default security context(s) attached. *The default session can be obtained by passing `true` to the session `CONSTRUCTOR`.*

Code Example

```
1  // Example in C++:
2
3  // create a file object in a specific session:
4  saga::file f1 (session, url);
5
6  // create a file object in the default session:
7  saga::file f2 (url);
```

SAGA objects created from *other SAGA objects inherit its* *another SAGA object inherit its* session, such as, for example, `saga::streams` from `saga::stream_server`. Only some objects do not need a session **at** creation time, and can hence be shared between sessions. These include:

saga::exception
saga::buffer
saga::iovec
saga::parameter
saga::context
saga::job_description
saga::metric
saga::exception
saga::task
saga::task_container

Note that tasks have no explicit session attached. The `saga::object` the task was created from, however, has a `saga::session` attached, *and that session instance is indirectly available, as the application can obtain that object via the* `get_object` *method call on the respective task instance.*

Multiple sessions can co-exist. *A single session can be shared between threads.*

If a `saga::session` object instance gets destroyed, or goes out of scope, the objects associated with that session survive. The implementation MUST ensure that the session is internally kept alive until the last *of that sessions objects* *object of that session* gets destroyed.

If the session object **instance** itself gets destroyed, the resources associated with that session MUST be freed immediately as the last object associated with that session gets destroyed. *The lifetime of the default session is, however, only limited by the lifetime of the SAGA application itself (see Notes about life time management in Section 2.5.3).*

Objects associated with different sessions MUST NOT influence each other in any way - for all practical purposes, they can be considered to be running in different application instances.

Any SAGA operation CAN throw an TIncorrectSession exception if it involves two different session handles.

Instances of the `saga::context` class (which encapsulates security information in SAGA) can be attached to a `saga::session` instance. The context instances are to be used by that session for authentication and authorization to *the* the backends used.

If a `saga::context` gets removed from a session, but that context is already/still used by any object created in that session, the context MAY continue to be used by these objects, and by objects which inherit the session from these objects, but not by any other objects. However, a call to `list_contexts` MUST NOT list the removed context after it *gets got* removed.

For the default ***session*** *instance, of any explicitly attached saga::context instances,* the list returned by a call to `list_contexts()` MUST include the default `saga::context` instances. *These are those contexts that are added to any* ***saga::session*** *by default, e.g. because they are picked up by the SAGA implementation from the application's run time environment. An application can, however, subsequently remove default contexts from the default session. A new, non-default session has initially no contexts attached.*

A SAGA implementation MUST document **which** default context instances it may create and attach to a saga::session. That set MAY change during runtime, but *SHOULD NOT* be changed once a `saga::session` instance was created. **For example**, two `saga::session` instances might have different default `saga::context` instances attached. Both sessions, however, will have these attached for their complete lifetime – *unless they expire or get otherwise invalidated*.

Default `saga::context` instances on a session can be removed from a session, with a call to `remove_context()`. That may result in a session with no contexts attached. *That session is still valid, but likely to fail on most autorization points.*

```
    - CONSTRUCTOR
      Purpose:  create the object
!     Format:   CONSTRUCTOR          (in  bool    default = true,
                                      out session obj)
+     Inputs:   default:             indicates if the default
+                                    session is returned
+     InOuts:   -
      Outputs:  obj:                 the newly created object
+     PreCond:  -
+     PostCond: -
+     Perms:    -
```

```
+     Throws:   NotImplemented
+               NoSuccess
      Notes:    - the created session has no context
                  instances attached.
+               - if 'default' is specified as 'true', the
+                 constructor returns a shallow copy of the
+                 default session, with all the default
+                 contexts attached.  The application can then
+                 change the properties of the default session,
+                 which is continued to be implicetly used on
+                 the creation of all saga objects, unless
+                 specified otherwise.


    - DESTRUCTOR
      Purpose:  destroy the object
      Format:   DESTRUCTOR           (in  session obj)
      Inputs:   obj:                  the object to destroy
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - See notes about lifetime management
+                 in Section 2
+     Perms:    -
      Throws:   -
+     Notes:    -


    - add_context
      Purpose:  attach a security context to a session
!     Format:   add_context          (in context  c);
!     Inputs:   c:                    Security context to add
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - the added context is deep copied, and no
+                 state is shared.
+               - any object within that session can use the
+                 context, even if it was created before
+                 add_context was called.
+     Perms:    -
+     Throws:   NotImplemented
+     Notes:    - if the session already has a context attached
!                which has exactly the same set of attribute
                 values as the parameter context, no action is
                 taken.
```

```
    - remove_context
      Purpose:  detach a security context from a session
!     Format:   remove_context       (in context  c);
!     Inputs:   c:                    Security context to remove
+     InOuts:   -
      Outputs:  -
+     Throws:   NotImplemented
                DoesNotExist
+     PreCond:  - a context with completely identical attributes
+                 is available in the session.
+     PostCond: - that context is removed from the session, and
+                 can from now on not be used by any object in
+                 that session, even if it was created before
+                 remove_context was called.
+     Perms:    -
!     Notes:    - this methods removes the context on the
!                 session which has exactly the same set of
                  parameter values as the parameter context.
                - a 'DoesNotExist' exception is thrown if no
                  context exist on the session which has the
                  same attributes as the parameter context.

    - list_contexts
      Purpose:  retrieve all contexts attached to a session
      Format:   list_contexts        (out array<context>
                                                contexts);
      Inputs:   -
+     InOuts:   -
!     Outputs:  contexts:             list of contexts of this
                                      session
+     PreCond:  -
+     PostCond: -
+     Perms:    -
+     Throws:   NotImplemented
      Notes:    - a empty list is returned if no context is
                  attached.
                - contexts may get added to a session by
                  default, hence the returned list MAY be
                  non-empty even if add_context() was never
                  called before.
                - a context might still be in use even if not
                  included in the returned list.  See notes
                  about context life time above.
```

3.5.3 Examples

Code Example

```
// c++ example
saga::session s;
saga::context c (saga::context::X509);

s.add_context   (c);

saga::directory  d (s, "gsiftp://remote.net/tmp/");
saga::file       f = d.open ("data.txt");

// file has same session attached as dir,
// and can use the same contexts
```

Code Example

```
// c++ example
saga::task    t;
saga::session s;

{
  saga::context c ("X509");

  s.add_context (c);

  saga::file f (s, url);

  t = f.copy <saga::task::Task> (target);

  s.remove_context (c);
}

// As it leaves the scope, the X509 context gets 'destroyed'.
// However, the copy task and the file object MAY continue to
// use the context, as its destruction is actually delayed
// until the last object using it gets destroyed.

t.run (); // can still use the context
```

3.6 SAGA Context Management

The saga::context class provides the functionality of a security information container. A context is 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. A single saga::context instance can be shared between threads and sessions. SAGA objects created from other SAGA objects inherit its session and thus also its context(s). Section 3.5 contains more information about the saga::session class, and also about the management and lifetime of saga::context instances associated with a SAGA session.

An implementation CAN implement various types of contexts, but MUST implement at least one type. The type of a saga::context instance to be created is specified by an enum which is the only argument to the context constructor.

On contexts with type Unknown, other methods than get_type() should not be called – otherwise an IncorrectState exception MUST be thrown.

Every context has a specific set of attributes which can be set/get via the SAGA attributes interface. Exactly what attributes a context offers depends on its type. A context MUST raise an exception if attributes not corresponding to its type are accessed.

*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 threads and multiple sessions. SAGA objects created from other SAGA objects inherit its session and thus also its context(s). Section 3.5 contains more information about the **saga::session** class, and also about the management and lifetime of **saga::context** instances associated with a SAGA session.*

A typical usage scenario is:

Code Example

```
1  // context usage scenario in c++
2
3  saga::context c_1, c_2;
4
5  // c_1 will use a globus proxy.  Set the type to globus, pick
6  // up the default globus settings, and then identify the proxy
7  // to be used
8  c_1.set_attribute ("Type", "globus");
9  c_1.set_defaults  ();
10 c_1.set_attribute ("UserProxy", "/tmp/special_x509up_u500");
11
12 // c_2 will be used as ssh context, and will just pick up the
13 // public/private key from $HOME/.ssh
14 c_2.set_attribute ("Type", "ssh");
15 c_2.set_defaults ();
16
```

```
17  // a saga session gets created, and uses both contexts
18  saga::session s;
19  s.add_context (c_1);
20  s.add_context (c_2);
21
22  // a remote file in this session can now be accessed via
23  // gridftp or ssh
24  saga::file f (s, "any://remote.net/tmp/data.txt");
25  f.copy ("data.bak");
```

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

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

The lifetime of `saga::context` instances **is** defined by the lifetime of those `saga::session` instances **the contexts are** associated with, and of those SAGA objects which have been created in these sessions. For detailed information about lifetime management, see *the introduction Section* 2.5.3, and the description of the SAGA session class in Section 3.5.

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

For example, a SAGA application A creates a `saga::stream_server` *instance. A SAGA application B creates a 'globus' type context, and, with a session using that context, creates a* `saga::stream` *instance connecting to the stream server of A. A should then obtain a context upon connection accept (see Sections on Monitoring, 3.9, and Streams, 4.5, for details). That context should then also have the type 'globus', its 'RemoteID' attribute should contain the distinguished name of the user certificate, and its attributes 'RemoteHost' and 'RemotePort' should have the appropriate values.*

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

3.6.1 Specification

```
  package saga.context
  {
-   enum context_type
-   {
-     Unknown         = -1,
-     Globus          =  1, // Globus
-     MyProxy         =  2, // MyProxy
-     SAML            =  3, // SAML
-     Unicore         =  4  // Unicore
```

```
-     SSH             =  5, // SSH
-     Kerberos        =  6, // Kerberos
-     UserPass        =  7  // FTP etc.
-   }
-
    class context : implements   saga::object
                    implements   saga::attributes
                // from object  saga::error_handler
    {
-     CONSTRUCTOR      (in  context_type type,
+     CONSTRUCTOR      (in  string       type = "",
+                       out context      obj);
!     DESTRUCTOR       (in  context      obj);

-     get_ctype        (out context_type type);
+     set_defaults     (void);

      // Attributes:
!     //   name:  Type
!     //   desc:  type of context
!     //   mode:  ReadWrite
!     //   type:  String
!     //   value: naming conventions as described above apply
!     //
!     //   name:  Server
!     //   desc:  server which manages the context
!     //   mode:  ReadWrite
!     //   type:  String
!     //   value: -
!     //   note:  - a typical example would be the contact
!     //            information for a MyProxy server, such as
!     //            'myproxy.remote.net:7512', for a 'myproxy'
!     //            type context.
!     //
!     //   name:  CertRepository
!     //   desc:  location of certificates and CA signatures
!     //   mode:  ReadWrite
!     //   type:  String
!     //   value: -
!     //   note:  - a typical example for a globus type context
!     //            would be "/etc/grid-security/certificates/".
!     //
!     //   name:  UserProxy
!     //   desc:  location of an existing certificate proxy to
!     //          be used
!     //   mode:  ReadWrite
```

```
!     //   type:  String
!     //   value: -
!     //   note:  - a typical example for a globus type context
!     //            would be "/tmp/x509up_u<uid>".
!     //
!     //   name:  UserCert
!     //   desc:  location of a user certificate to use
!     //   mode:  ReadWrite
!     //   type:  String
!     //   value: -
!     //   note:  - a typical example for a globus type context
!     //            would be "$HOME/.globus/usercert.pem".
!     //
!     //   name:  UserKey
!     //   desc:  location of a user key to use
!     //   mode:  ReadWrite
!     //   type:  String
!     //   value: -
!     //   note:  - a typical example for a globus type context
!     //            would be "$HOME/.globus/userkey.pem".
!     //
!     //   name:  UserID
!     //   desc:  user id or user name to use
!     //   mode:  ReadWrite
!     //   type:  String
!     //   value: -
!     //   note:  - a typical example for a ftp type context
!     //            would be "anonymous".
!     //
!     //   name:  UserPass
!     //   desc:  password to use
!     //   mode:  ReadWrite
!     //   type:  String
!     //   value: -
!     //   note:  - a typical example for a ftp type context
!     //            would be "anonymous@localhost".
!     //
!     //   name:  UserVO
!     //   desc:  the VO the context belongs to
!     //   mode:  ReadWrite
!     //   type:  String
!     //   value: -
!     //   note:  - a typical example for a globus type context
!     //            would be "O=dutchgrid".
!     //
!     //   name:  LifeTime
```

```
!     //   desc:  time up to which this context is valid
!     //   mode:  ReadWrite
!     //   type:  Int
!     //   value: -1
!     //   note:  - format: time and date specified in number of
!     //            seconds since epoch
!     //          - a value of -1 indicates an infinit lifetime.
!     //
!     //   name:  RemoteID
!     //   desc:  user ID for an remote user, who is identified
!     //          by this context.
!     //   mode:  ReadOnly
!     //   type:  String
!     //   value: -
!     //   note:  - a typical example for a globus type context
!     //            would be
!     //            "/O=dutchgrid/O=users/O=vu/OU=cs/CN=Joe Doe".
!     //
!     //   name:  RemoteHost
!     //   desc:  the hostname where the connection origininates
!     //          which is identified by this context.
!     //   mode:  ReadOnly
!     //   type:  String
!     //   value: -
!     //
!     //   name:  RemotePort
!     //   desc:  the port used for the connection which is
!     //          identified by this context.
!     //   mode:  ReadOnly
!     //   type:  String
!     //   value: -
!     //
    }
  }
```

3.6.2 Specification Details

Class `context`

Following attributes MUST be supported by the corresponding context types, with examples for default values given in brackets, where appropriate:

```
-     Unknown:
-       No attributes supported
-
-     Globus:
-       ReadWrite:
-         UserProxy     (/tmp/x509up_u<uid>)
-         CertLocation  (/etc/grid-security/certificates/)
-         UserCert      ($HOME/.globus/usercert.pem)
-         UserKey       ($HOME/.globus/userkey.pem)
-         UserPass
-       ReadOnly:
-         RemoteID
-         RemoteHost
-         RemotePort
-
-
-     MyProxy:
-       ReadWrite:
-         Server        (localhost)
-         UserProxy     (/tmp/x509up_u<uid>)
-         UserMyProxy   (/tmp/myproxy-proxy.<uid>.<pid>)
-         CertLocation  (/etc/grid-security/certificates/)
-         UserCert      ($HOME/.globus/usercert.pem)
-         UserKey       ($HOME/.globus/userkey.pem)
-         UserPass
-       ReadOnly:
-         RemoteID
-         RemoteHost
-         RemotePort
-
-
-     SAML:
-       ReadWrite:
-         UserID
-         UserPass
-       ReadOnly:
-         RemoteID
-         RemoteHost
-         RemotePort
-
-
-     Unicore:
-       ReadWrite:
-         UserCert      ($HOME/.unicore/keystore)
-         UserPass
-       ReadOnly:
```

```
-         RemoteID
-         RemoteHost
-         RemotePort
-
-
-     SSH:
-       ReadWrite:
-         CertLocation  ($HOME/.ssh/)
-         UserCert      ($HOME/.ssh/id_dsa.pub)
-         UserPass      ($HOME/.ssh/id_dsa)
-       ReadOnly:
-         RemoteID
-         RemoteHost
-         RemotePort
-
-
-     Kerberos:
-       ReadWrite:
-         Server        (localhost)
-         UserID
-         UserPass
-       ReadOnly:
-         RemoteID
-         RemoteHost
-         RemotePort
-
-
-     UserPass:
-       ReadWrite:
-         UserID
-         UserPass
-       ReadOnly:
-         RemoteID
-         RemoteHost
-         RemotePort
-
-   Other context types MAY be specified by a SAGA
-   implementation.
-
-
-   - CONSTRUCTOR
-     Purpose:  create a security context
-     Format:   CONSTRUCTOR          (in  context_type type,
-                                     out context      obj);
-     Inputs:   type:                 type of context
-     InOuts:   -
```

```
-     Outputs:  obj:                  the newly created object
-     PreCond:  -
-     PostCond: -
-     Perms:    -
-     Throws:   BadParameter
-     Notes:    - BadParameter is thrown if a context type is
-                 not supported (NOT NotImplemented).
-

+   - CONSTRUCTOR
+     Purpose:  create a security context
+     Format:   CONSTRUCTOR          (in  stringt type = "",
+                                     out context obj);
+     Inputs:   type:                 initial type of context
+     InOuts:   -
+     Outputs:  obj:                  the newly created object
+     PreCond:  -
+     PostCond: -
+     Perms:    -
+     Throws:   NotImplemented
+               IncorrectState
+               Timeout
+               NoSuccess
+     Notes:    - if type is given (i.e. non-empty), then the
+                 CONSTRUCTOR internally calls set_defaults().
+                 The notes to set_defaults apply.

    - DESTRUCTOR
      Purpose:  destroy a security context
!     Format:   DESTRUCTOR           (in context obj);
!     Inputs:   obj:                  the object to destroy
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - See notes about lifetime management
+                 in Section 2
+     Perms:    -
      Throws:   -
+     Notes:    -

-
-   - get_context_type
-     Purpose:  query the context type
-     Format:   get_context_type     (out context_type type);
-     Inputs:   -
-     InOuts:   -
```

```
-     Outputs:  type:                 type of context
-     PreCond:  -
-     PostCond: -
-     Perms:    -
-     Throws:   -
-     Notes:    -
-
+   - set_defaults
+     Purpose:  set default values for specified context type
+     Format:   set_defaults         (void);
+     Inputs:   -
+     InOuts:   -
+     Outputs:  -
+     PreCond:  -
+     PostCond: - the context is valid, and can be used for
+                 authorization.
+     Perms:    -
+     Throws:   NotImplemented
+               IncorrectState
+               Timeout
+               NoSuccess
+     Notes:    - the method avaluates the value of the 'Type'
+                 attribute, and of all other non-empty
+                 attributes, and, based on that information,
+                 tries to set sensible default values for all
+                 previously empty attributes.
+               - if the 'Type' attribute has an empty value, an
+                 'IncorrectState' exception is thrown.
+               - this method can be called more than once on
+                 a context instance.
+               - if the implementation cannot create valid
+                 default values based on the available
+                 information, an 'NoSuccess' exception is
+                 thrown, and a detailed error message is given,
+                 describing why no default values could be
+                 set.
```

3.7 SAGA Permission Model

This whole section has been added to the specification!

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 packages 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.

A general problem with this approach is that it is difficult to anticipate how users and user groups are identified by various grid middleware systems. In particular, any translation of permissions specified for one grid middleware is likely not completely translatable to permissions for another grid middleware.

For example, assume that a `saga::file` instance gets created via ssh, and permissions are set for the file to be readable and executable by a specific POSIX user group ID. Which implications do these permissions have with respect to operations performed with GridFTP, using a Globus certificate? The used X509 certificates have (a) no notion of groups (groups are implicit due to the mapping of the `grid-mapfile`), and (b) are not mappable to group ids; and (c) GridFTP ignores the executable flag on files.

For this reason, it is anticipated that the permission model described in this section has the following, undesired consequences and limitations:

- Applications using this interface are not expected to be fully portable between different SAGA implementations. (In cases like having two SAGA implementations that use different middleware backends for accessing the same resources.)
- A SAGA implementation MUST document which permission it supports, for which operations.
- A SAGA implementation MUST document if it supports group level permissions.
- A SAGA implementation MUST document how user and group IDs are to be formed.

Note that there are no separate calls to get/set user, group and world permissions: this information must be part of the IDs the methods operate upon. To set/get permisions for 'world' (i.e. anybody), the ID `'*'` is used.

IDs

SAGA can not, by design, define globally unique identifiers in a portable way. For example, it would be impossible to map, transparently and bi-directionally, a Unix user ID and an associated X509 Distinguished Name on any resource onto the same hypothetical SAGA user ID, at least not without explicit support by the grid middleware (e.g., by having access to the Globus `grid-mapfile`). That support is, however, rarely available.

It is thus required that SAGA implementations MUST specify how the user and group IDs are formed that they support. In general, IDs which are valid for the `UserID` attribute of the SAGA `context` instances SHOULD also be valid IDs to be used in the SAGA permission model.

A typical usage scenario is (extended from the context usage scenario):

Code Example

```
1  // context and permission usage scenario in C++
2
3  saga::context c_1 ("globus")
4  saga::context c_2 ("ssh");
5
6  // c_1 is a globus proxy.  Identify the proxy to be used,
7  // and pick up the other default globus settings
8  c_1.set_attribute ("UserProxy", "/tmp/special_x509up_u500");
9  c_1.set_defaults  ();
10
11 // c_2 is a ssh context, and will just pick up the
12 // public/private key from $HOME/.ssh
13 c_2.set_defaults ();
14
15 // a saga session gets created, and uses both contexts
16 saga::session s;
17 s.add_context (c_1);
18 s.add_context (c_2);
19
20 // a remote file in this session can now be accessed via
21 // gridftp or ssh
22 saga::file f (s, "any://remote.net/tmp/data.txt");
23 f.copy ("data.bak");
24
25 // write permissions can be set for both context IDs
26 f.permission_allow (c_1.get_attribute ("UserID"), Write);
27 f.permission_allow (c_2.get_attribute ("UserID"), Write);
```

For middleware systems where group and user ids can clash, the IDs should be

implemented as `'user-<id>'` and `'group-<id>'`. For example: on Unix, the name 'mail' can (and often does) refer to a user and a group. In that case, the IDs should be expressed as `'user-mail'` and `'group-mail'`, respectively. The ID `'*'` is always reserved, as described above.

Permissions for a user ID supersede the permissions for a group ID, which supersede the permissions for `'*'` (all). If a user is in multiple groups, and the group's permissions differ, the most permissive permission applies.

3.7.1 Permissions for Multiple Backends

In SAGA, an entity which provides the `permissions` interface always has exactly one owner, for one middleware backend. However, this implies that for SAGA implementations with multiple backend bindings, multiple owner IDs may be valid. For example, `"/O=dutchgrid/O=users/O=vu/OU=cs/CN=Joe Doe"` and `"user-jdoe"` might be equally valid IDs, at the same time, if the implementation supports local Unix access and GridFTP access to a local file. As long as the ID spaces do not conflict, the `permissions` interface obviously allows to set permissions individually for both backends. In case of conflicts, the application would need to create new SAGA objects from sessions that contain only a single context, representing the desired backend's security credentials. As such situations are considered to be very rare exceptions in the known SAGA use cases, we find this limitation accetable.

Note that, for SAGA implementations supporting multiple middleware backends, the `permissions` interface can operate on permissions for any of these backends, not only for the one that was used by the original creation of the object instance. Such a restriction would basically inhibit implementations with dynamic ("late") binding to backends.

Conflicting Backend Permission Models

Some middleware backends may not support the full range of permissions, e.g., they might not distinguish between `Query` and `Read` permissions. A SAGA implementation MUST document which permissions are supported. Trying to set an unsupported permission reults in a `BadParameter` exception, and NOT in a `NotImplemented` exception – that would indicate that the method is not available at all, i.e. that no permission model at all is available for this particular implementation.

An implementation MUST NOT silently merge permissions, according to its own model – that would break for example the following code:

```
file.permissions_allow ("user-jdoe", Query);
file.permissions_deny  ("user-jdoe", Read );
off_t file_size = file.get_size ();
```

If an implementation binds to a system with standard Unix permissions and does not throw a `BadParameter` exception on the first call, but silently sets `Read` permissions instead, because that does also allow query style operations on Unix, then the code in line three would fail for no obvious reason, because the second line would revoke the permissions from line one.

Initial Permission Settings

If new grid entities get created via the SAGA API, the owner of the object is set to the value of the `'UserID'` attribute of the context used during the creation. Note that for SAGA implementations with support for multiple middleware backends, and support for late binding, this may imply that the owner is set individually for one, some or all of the supported backends.

Creating grid entities may require specific permissions on other entities. For example:

- file creation requires `Write` permissions on the parent directory.
- executing a file requires `Read` permissions on the parent directory.

An implementation CAN set initial permissions other than `Owner`. An implementation SHOULD document which initial permission settings an application can expect.

The specification of the `ReadOnly` flag on the creation or opening of SAGA object instances, such as `saga::file` instances, causes the implementation to behave as if the `Write` permission on the entity on that instance is not available, even if it is, in reality, available. The same holds for the `WriteOnly` flag and the availability of the `Read` permission on that entity.

Permission Definitions in the SAGA specification

The SAGA specification normatively defines for each operation, which permissions are required for that operation. If a permission is supported, but not set, the method invocation MUST cause a `PermissionDenied` exception. An implementation MUST document any deviation from this scheme, e.g., if a specified

permission is not supported at all, or cannot be tested for a particular method. An example of such a definition is (from the `monitorable` interface):

```
- list_metrics
  Purpose:  list all metrics associated with the object
  Format:   list_metrics       (out array<string>   names);
  Inputs:   -
  InOuts:   -
  Outputs:  names:              array of names identifying
                                the metrics associated with
                                the object instance
  PreCond:  -
  PostCond: -
  Perms:    Query
  Throws:   NotImplemented
            PermissionDenied
            AuthorizationFailed
            AuthenticationFailed
            Timeout
            NoSuccess
  Notes:    - [...]
```

This example implies that for the session in which the `list_metrics()` operation gets performed, there must be at least one context for which's attribute `'UserID'` the `Query` permission is both supported and available; otherwise, the method MUST throw a `PermissionDenied` exception. If `Query` is not supported by any of the backends for which a context exists, the implementation MAY try the backends to perform the operation anyway.

For some parts of the specification, namely for attributes and metrics, the `mode` specification is normative for the respective, required permission. For example, the mode attribute `ReadOnly` implies that a `Write` permission, required to change the attribute, is never available.

The `PermissionDenied` exception in SAGA

SAGA supports a `PermissionDenied` exception, as documented in Section 3.1. This exception can originate from various circumstances, that are not necessarily related to the SAGA permission model as described here. However, if the reason why that exception is raised maps onto the SAGA permission model, the exception's error message MUST have the following format (line breaks added for readability):

```
PermissionDenied: no <PERM> permission
                  on <ENTITY> <NAME>
                  for <ID>
```

Here, `<PERM>` denotes which permission is missing, `<ENTITY>` denotes on what kind of entity this permission is missing. `<NAME>` denotes which entity misses that permission, and `<ID>` denotes which user is missing that permission.

`<PERM>` is the literal string of the `permission` enum defined in this section. `<ENTITY>` is the type of backend entity which is missing the permission, e.g. `file`, `directory`, `job_service` etc. Whenever possible, the literal class name of the respective SAGA class name SHOULD be used. `<NAME>` SHOULD be a URL or literal name which allows the end user to uniquely identify the entity in question. `<ID>` is the value of the `UserID` attribute of the context used for the operation (the notes about user IDs earlier in this section apply).

Some examples for complete error messages are:

```
PermissionDenied: no Read permission
                  on file http:////tmp/test.dat
                  for user-jdoe

PermissionDenied: no Write permission
                  on directory http:////tmp/
                  for user-jdoe

PermissionDenied: no Query permission
                  on logical_file rls:////tmp/test
                  for /O=ca/O=users/O=org/CN=Joe Doe

PermissionDenied: no Query permission
                  on job [fork://localhost]-[1234]
                  for user-jdoe

PermissionDenied: no Exec permission
                  on RPC [rpc://host/matmult] for
                  for /O=ca/O=users/O=org/CN=Joe Doe
```

Note to users

The description of the SAGA permission model above should have made clear that, in particular, the support for multiple backends makes it difficult to strictly enforce the permissions specified on application level. Until a standard for permission management for Grid application emerges, this situation is unlikely to change. Applications should thus be careful to trust permissions specified in SAGA, and should ensure to use an implementation which fully supports

and enforces the permission model, e.g., they should choose an implementation which binds to a single backend.

3.7.2 Specification

```
package saga.permissions
{
  enum permission
  {
    None      =  0,
    Query     =  1,
    Read      =  2,
    Write     =  4,
    Exec      =  8,
    Owner     = 16,
    All       = 31
  }

  interface permissions : implements saga::async
  {
    // setter / getters
    permissions_allow       (in  string          id,
                             in  int             perm);
    permissions_deny        (in  string          id,
                             in  int             perm);
    permissions_check       (in  string          id,
                             in  int             perm,
                             out bool            value);
    get_owner               (out string          owner);
    get_group               (out string          group);
  }
}
```

3.7.3 Specification Details

Enum `permission`

This enum specifies the available permissions in SAGA. The following examples demonstrate which type of operations are allowed for certain permissions, and which aren't. To keep these examples concise, they are chosen from the following

list, with the convention that those operations in this list, which are not listed in the respective example section, are *not* allowed for that permission. In general, the availability of one permission does not imply the availability of any other permission (with the exception of `Owner`, as described below).

- provide information about a metric, and its properties
- provide information about file size, access time and ownership
- provide information about job description, ownership, and runtime
- provide information about logical file access time and ownership
- provide access to a job's I/O streams
- provide access to the list of replicas of a logical file
- provide access to the contents of a file
- provide access to the value of a metric
- provide means to change the ownership of a file or job
- provide means to change the permissions of a file or job
- provide means to fire a metric
- provide means to connect to a stream server
- provide means to manage the entries in a directory
- provide means to manipulate a file or its meta data
- provide means to manipulate a job's execution or meta data
- provide means to manipulate the list of replicas of a logical file
- provide means to run an executable

The following permissions are available in SAGA:

`Query`

This permission identifies the ability to *access all meta data of an entity*, and thus to obtain any information about an entity. If that permission is not available for an actor, that actor MUST NOT be able to obtain any information about the queried entity, if possible not even about its existence. If that permission is available for an actor, the actor MUST be able to query for any meta data on the object which (a) do imply changes on the entities state, and (b) are part of the *content* of the entity (i.e., do not comprise its data).

Note that for logical files, attributes are part of the data of the entities (i.e., the meta data belong to the logical file's data).

An authorized `Query` operation can:

- provide information about a metric, and its properties
- provide information about file size, access time and ownership

- provide information about job description, ownership, and runtime
- provide information about logical file access time and ownership

`Read`

This permission identifies the ability to *access the contents and the output of an entity*. If that permission is not available for an actor, that actor MUST NOT be able to access the data of the entity. That permission does not imply the authorization to change these data, or to manipulate the entity. That permission does also not imply `Query` permissions, i.e. the permission to access the entity's meta data.

An authorized `READ` operation can:

- provide access to a job's I/O streams
- provide access to the list of replicas of a logical file
- provide access to the contents of a file
- provide access to the value of a metric

`Write`

This permission identifies the ability to *manipulate the contents of an entity*. If that permission is not available for an actor, that actor MUST NOT be able to change neither data nor meta data of the entity. That permission does not imply the authorization to read these data of the entity, nor to manipulate the entity. That permission does also not imply `Query` permissions, i.e., the permission to access the entity's meta data.

Note that, for a directory, its entries comprise its data. Thus, `Write` permissions on a directory allow to manipulate all entries in that directory – but do not imply the ability to change the data of these entries. For example, `Write` permissions on the directory `'/tmp'` allows to move `'/tmp/a'` to `'/tmp/b'`, or to remove these entries, but does not imply the ability to perform a `read()` operation on `'/tmp/a'`.

An authorized `Write` operation can:

- provide means to manage the entries in a directory
- provide means to manipulate a file or its meta data
- provide means to manipulate a job's execution or meta data
- provide means to manipulate the list of replicas of a logical file

`Exec`

This permission identifies the ability to *perform an action on an entity*. If that permission is not available for an actor, that actor MUST NOT be able to perform that action. The actions covered by that permission are usually those which affect the state of the entity, or which create a new entity.

An authorized `Exec` operation can:

- provide means to fire a metric

- provide means to connect to a stream server
- provide means to run an executable

`Owner`

This permission identifies the ability to *change permissions and ownership of an entity*. If that permission is not available for an actor, that actor MUST NOT be able to change any permissions or the ownership of an entity. As this permission indirectly implies full control over all other permissions, it does also imply that an actor with that permission can perform *any* operation on the entity. `Owner` is not listed as additional required permission in the specification details for the individual methods, but only listed for those methods, where `Owner` is an explicit permission requirement which cannot be replaced by any other permission.

An authorized `Owner` operation can:

- provide means to change the ownership of a file or job
- provide means to change the permissions of a file or job
- perform *any* other operation, including all operations from the original list of examples above

Note that only one user can own an entity. For example, the following sequence:

```
file.permissions_allow ("Tarzan", saga::permission::Owner);
file.permissions_allow ("Jane",   saga::permission::Owner);
```

would result in a file ownership by `'Jane'`.

Also note that

```
file.permissions_allow ("*", saga::permission::Owner);
```

or

```
file.permissions_deny (id, saga::permission::Owner);
```

will never be possible, and will throw a `BadParameter` exception.

Interface `permissions`

```
- permissions_allow
  Purpose:  enable permission flags
  Format:   permissions_allow    (in  string     id,
                                  in  int        perm);
  Inputs:   id:                  id to set permission for
            perm:                permissions to enable
  InOuts:   -
  Outputs:  -
```

```
  PreCond:  -
  PostCond: - the permissions are enabled.
  Perms:    Owner
  Throws:   NotImplemented
            BadParameter
            PermissionDenied
            AuthorizationFailed
            AuthenticationFailed
            Timeout
            NoSuccess
  Notes:    - an id '*' sets the permissions for all (world)
            - whether an id is interpreted as a group id is up to
              the implementation.  An implementation MUST
              specify how user and group id's are formed.
            - the 'Owner' permission can not be set to the
              id '*' (all).
            - if the given id is unknown or not supported, a
              'BadParameter' exception is thrown.


- permissions_deny
  Purpose:  disable permission flags
  Format:   permissions_deny     (in  string     id,
                                  in  int        perm);
  Inputs:   id:                  id to set permissions for
            perm:                permissions to disable
  InOuts:   -
  Outputs:  -
  PreCond:  -
  PostCond: - the permissions are disabled.
  Perms:    Owner
  Throws:   NotImplemented
            BadParameter
            PermissionDenied
            AuthorizationFailed
            AuthenticationFailed
            Timeout
            NoSuccess
  Notes:    - an id '*' sets the permissions for all (world)
            - whether an id is interpreted as a group id is up to
              the implementation.  An implementation MUST
              specify how user and group id's are formed.
            - the 'Owner' permission can not be set to the
              id '*' (all).
            - if the given id is unknown or not supported, a
              'BadParameter' exception is thrown.
```

```
- permissions_check
  Purpose:  check permission flags
  Format:   permissions_check    (in  string     id,
                                  in  int        perm,
                                  out bool       allow);
  Inputs:   id:                   id to check permissions for
            perm:                 permissions to check
  InOuts:   -
  Outputs:  allow:                indicates if, for that id,
                                  the permissions are granted
                                  (true) or not.
  PreCond:  -
  PostCond: -
  Perms:    Query
  Throws:   NotImplemented
            BadParameter
            PermissionDenied
            AuthorizationFailed
            AuthenticationFailed
            Timeout
            NoSuccess
  Notes:    - an id '*' gets the permissions for all (world)
            - 'true' is only returned when all permissions
              specified in 'perm' are set for the given id.
            - if the given id is unknown or not supported, a
              'BadParameter' exception is thrown.


- get_owner
  Purpose:  get the owner of the entity
  Format:   get_owner            (out string     owner);
  Inputs:   -
  InOuts:   -
  Outputs:  owner:                id of the owner
  PreCond:  -
  PostCond: -
  Perms:    Query
  Throws:   NotImplemented
            PermissionDenied
            AuthorizationFailed
            AuthenticationFailed
            Timeout
            NoSuccess
  Notes:    - returns the id of the owner of the entity
```

```
                - an entity, on which the permission interface is
                  available, always has exactly one owner: this
                  method MUST NOT return an empty string, and
                  MUST NOT return '*' (all), and MUST NOT return
                  a group id.


  - get_group
    Purpose:  get the group owning the entity
    Format:   get_group            (out string     group);
    Inputs:   -
    InOuts:   -
    Outputs:  group:                id of the group
    PreCond:  -
    PostCond: -
    Perms:    Query
    Throws:   NotImplemented
              PermissionDenied
              AuthorizationFailed
              AuthenticationFailed
              Timeout
              NoSuccess
    Notes:    - returns the id of the group owning the entity
              - this method MUST NOT return '*' (all), and
                MUST NOT return a user id.
              - if the implementation does not support groups,
                the method returns an empty string.
```

3.7.4 Examples

Code Example

```
1
2   // c++ example
3   {
4     // create a file in the default session
5     saga::file f (url, saga::file::Create
6                      | saga::file::Exclusive):
7
8     // get all contexts of the default session, and for each...
9     std::list <saga::context> ctxs = theSession.list_contexts ();
10
11    for ( int i = 0; i < ctxs.size (); i++ )
12    {
13      saga::context ctx = ctxs[i];
```

```
14
15      // set the file to be executable
16      f.permission_allow (ctx.get_attribute ("UserID"),
17                          saga::permission::Exec);
18    }
19
20    // the file should now be usable for job submission for all
21    // contexts in the default session.  Often, however, only
22    // one context will succeed in setting the permission: the
23    // one which was used for creation in the first place.  In
24    // that case, job submission is most likely to succeed with
25    // that context, too.
26  }
```

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 interface **name** `attributes` is somewhat misleading: it seems to imply that an object implementing this interface `IS-A` set of attributes. What we actually mean is that an object implementing this interface `HAS` attributes. *For want* *In the absence* of a better name, we left it `attributes`, but **implementors** and users should be aware of the actual meaning (**the** proper interface **name** would be 'attributable', which sounds awkward).

The SAGA *specification* defines attributes which MUST be supported by the various SAGA objects, and *also defines* their default values, and *also defines* those which CAN be supported. An implementation MUST motivate and document if a specified attribute is not supported.

3.8.1 Specification

```
  package saga.attributes
  {
    interface attributes
    {
      // setter / getters
      set_attribute           (in  string          key,
                               in  string          value);
      get_attribute           (in  string          key,
                               out string          value);
      set_vector_attribute    (in  string          key,
                               in  array<string>   values);
      get_vector_attribute    (in  string          key,
                               out array<string>   values);
      remove_attribute        (in  string          key);

      // inspection methods
      list_attributes         (out array<string>   keys);
!     find_attributes         (in  array<string>   pattern,
```

```
-                              in  string          val_pattern,
                               out array<string>   keys);
      attribute_is_readonly   (in  string          key,
                               out bool            test);
      attribute_is_writable   (in  string          key,
                               out bool            test);
      attribute_is_removable  (in  string          key,
                               out bool            test);
      attribute_is_vector     (in  string          key,
                               out bool            test);
-     attribute_equals        (in  string          key,
-                              in  string          value,
-                              out bool            test);
    }
  }
```

3.8.2 Specification Details

The `attributes` interface in SAGA provides a uniform paradigm to set and query parameters and properties of SAGA objects. Although the `attributes` interface is generic by design (i.e. it allows arbitrary keys and values to be used), its use in SAGA is mostly limited to a finite and well defined set of keys.

In several languages, attributes can much more elegantly *be* expressed by native means - e.g. by using hash tables in Perl. Bindings for such languages MAY allow to use a native interface *additionally* to the one described here.

Several SAGA objects have very frequently used attributes. To simplify usage of these objects, setter and getter methods MAY be defined by the various language bindings, again *additionally* to the interface described below. For attributes of native non-string types, these setter/getters MAY be typed.

For example, additionally to:

```
stream.set_attribute ("BufferSize", "1024");
```

a language binding might allow:

```
stream.set_buffer_size (1024);  // int type
```

Further, in order to limit semantic and syntactic ambiguities (e.g., due to spelling deviations), language bindings MUST define known attribute keys as constants, such as (in C):

```
#define SAGA_BUFFERSIZE "BufferSize"
...
stream.set_attribute (SAGA_BUFFERSIZE, "1024");
```

The distinction between scalar and vector attributes is *somewhat artificial, and is* supposed to help those languages where *that nature* *this aspect* of attributes cannot be handled transparently, e.g. by overloading. Bindings for languages such as Python, Perl and C++ CAN hide **this** distinction as long as both access types are supported.

To simplify handling of scalar/vector attributes, vector attributes can be specified as comma delimited strings (leading space after comma is ignored, unless escaped):

val 1: "home, sweet home"

val 2: "Open GF"

val 3: " SAGA"

string: "home sweet home, Open GF, SAGA"

That format is returned if scalar getters are used for vector attributes, and can be used for scalar setters for vector attributes. Vector setters/getters handle scalar attributes as vectors of length one.

The order of the elements of vector attributes is well defined, and *Elements of vector attributes are ordered. This order* MUST be preserved by the SAGA implementation. **Comparison** also relies on ordering (i.e. `'one two'` does not equal `'two one'`). For example, **this** order is significant for the `saga::job_description` attribute `'Arguments'`, which represents command line arguments for a job.

Attributes are expressed as string values. They have, *however*, a type, which defines the formatting of that string. The allowed types are `String`, `Int`, `Enum`, `Float`, `Bool`, and `Time` (the same as metric value types). Additionally, **attributes** are qualified as either `Scalar` or `Vector`. The default is `Scalar`.

Values of `String` type attributes are expressed as-is. *however, comma, backslashes and leading spaces need to be escaped by a backslash, as described above.*

Values of `Int` (i.e. Integer) type attributes are expressed as they would in result of a printf of the format *`'%lld'`*, as defined by POSIX.

Values of `Enum` type attributes are expressed as strings, and have the literal value of the respective enums as defined in this document. For example, the initial task states would have the values `'New'`, `'Running'` and `'Done'`.

Values of `Float` (*i.e. floating* point) type attributes are expressed as they would in result of a printf of the format *`'%Lf'`*, as defined by POSIX.

Values of `Bool` type attributes MUST be expressed as 'True' or 'False'.

Values of `Time` type attributes MUST be expressed as they would in result of a call to `ctime()`, as defined by POSIX. Applications can also specify these attribute values as seconds since epoch (this formats the string as an `Int` type), but all time attributes set by the implementation MUST be in `ctime()` format. Applications should be aware of the `strptime()` and `strftime()` methods defined in POSIX, which assist time conversions.

3.8.3 Attribute Definitions in the SAGA specification

The SAGA specification defines a number of attributes which MUST or CAN be supported, for various SAGA objects. An example *of* such a definition is (from the Metric object):

```
class metric ...
{
  ...

  // Attributes:
  //   name:  Name
  //   desc:  name of metric
  //   mode:  ReadOnly
  //   type:  String
  //   value: -
  //   notes: naming conventions as described below apply
  //
  //   ...
}
```

These specifications are NORMATIVE, even if described as comments in the SIDL specification! The specified attributes MUST be supported by an implementation, unless noted otherwise, as:

```
//  mode:  ReadOnly, optional
//  mode:  ReadWrite, optional
```

If an attribute MUST be supported, but the SAGA implementation cannot support that attribute, any set/get on that attribute MUST throw a `NotImplemented` exception, and the error message MUST state `"Attribute <name> not available in this implementation"`.

If the default value is *denoted as '–', then the* attribute *is, by default, not set at all.* *Non-optional attributes MUST have a default value (which can be an empty string).*

Attribute support can 'appear' and 'go away' during the lifetime of an object (e.g., as late binding implementations switch the backend). Any set on an attribute which got removed ('dead attribute') MUST throw a `DoesNotExist` exception. However, dead attributes MUST stay available for read access. The SAGA implementation MUST NOT change *that such an* attribute's value, as long as it is not available. Allowed values for mode are `ReadOnly` and `ReadWrite`.

It is not allowed to add attributes other than those specified in this document, unless explicitly allowed, as:

```
//  Attributes (extensible):
```

The `find_attributes()` method accepts *a list* of patterns, and returns a list of keys for those attributes which match any one of the specified patterns (`OR` semantics). The *allowed* patterns *describe both attribute keys and values*, and are *formatted as:*

```
<key-pattern>=<value-pattern>
```

Both the ***key-pattern*** *and the* ***value-pattern*** *can contain wildcards as defined in the description of the SAGA* ***namespace*** *package. If a* ***key-pattern*** *contains an '=' character, that character must be escaped by a backslash, as must any backslash character itself. The* ***value-pattern*** *can be empty, and the method will then return all attribute keys which match the* ***key-pattern****. The equal sign '=' can then be ommited from the pattern.*

Interface `attributes`

```
    - set_attribute
      Purpose:  set an attribute to a value
      Format:   set_attribute        (in string key,
                                      in string value);
      Inputs:   key:                 attribute key
                value:               value to set the
                                     attribute to
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: -
+     Perms:    Write
+     Throws:   NotImplemented
                BadParameter
                DoesNotExist
-               ReadOnly
```

```
+               IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
!     Notes:    - an empty string means to set an empty value
                  (the attribute is not removed).
                - the attribute is created, if it does not exist
                - a 'PermissionDenied' exception is thrown if the
                  attribute to be changed is ReadOnly.
                - only some SAGA objects allow to create new
                  attributes - others allow only access to
                  predefined attributes.  If a non-existing
                  attribute is queried on such objects, a
                  'DoesNotExist' exception is raised
                - changes of attributes may reflect changes of
                  endpoint entity properties.  As such,
!                 authorization and/or authentication may fail
                  for settings such attributes, for some
!                 backends.  In that case, the respective
                  'AuthenticationFailed', 'AuthorizationFailed',
                  and 'PermissionDenied' exceptions are thrown.
!                 For example, an implementation may forbid to
                  change the saga::stream 'Bufsize' attribute.
                - if an attribute is not well formatted, or
                  outside of some allowed range, a 'BadParameter'
                  exception with a descriptive error message is
                  thrown.
+               - if the operation is attempted on a vector
+                 attribute, an 'IncorrectState' exception is
+                 thrown.
                - setting of attributes may time out, or may fail
                  for other reasons - which causes a 'Timeout' or
                  'NoSuccess' exception, respectively.


    - get_attribute
!     Purpose:  get an attribute value
      Format:   get_attribute        (in  string key,
                                      out string value);
      Inputs:   key:                 attribute key
+     InOuts:   -
      Outputs:  value:               value of the attribute
+     PreCond:  -
+     PostCond: -
```

```
+     Perms:    Query
+     Throws:   NotImplemented
                DoesNotExist
+               IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
!     Notes:    - queries of attributes may imply queries of
                  endpoint entity properties.  As such,
!                 authorization and/or authentication may fail
                  for querying such attributes, for some
!                 backends.  In that case, the respective
                  'AuthenticationFailed', 'AuthorizationFailed',
                  and 'PermissionDenied' exceptions are thrown.
!                 For example, an implementation may forbid to
                  read the saga::stream 'Bufsize' attribute.
                - reading an attribute value for an attribute
                  which is not in the current set of attributes
                  causes a 'DoesNotExist' exception.
+               - if the operation is attempted on a vector
+                 attribute, an 'IncorrectState' exception is
+                 thrown.
                - getting attribute values may time out, or may
                  fail for other reasons - which causes a
                  'Timeout' or 'NoSuccess' exception,
                  respectively.


    - set_vector_attribute
      Purpose:  set an attribute to an array of values.
      Format:   set_vector_attribute (in string          key,
                                      in array<string>   values);
      Inputs:   key:                  attribute key
                values:               array of attribute values
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: -
+     Perms:    Write
+     Throws:   NotImplemented
                BadParameter
                DoesNotExist
-               ReadOnly
+               IncorrectState
```

```
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - the notes to the set_attribute() method apply.
+               - if the operation is attempted on a scalar
+                 attribute, an 'IncorrectState' exception is
+                 thrown.


    - get_vector_attribute
      Purpose:  get the array of values associated with an
                attribute
      Format:   get_vector_attribute (in string           key,
                                      out array<string>   values);
      Inputs:   key:                  attribute key
+     InOuts:   -
      Outputs:  values:               array of values of the
                                      attribute.
+     PreCond:  -
+     PostCond: -
+     Perms:    Query
+     Throws:   NotImplemented
                DoesNotExist
+               IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - the notes to the get_attribute() method apply.
+               - if the operation is attempted on a scalar
+                 attribute, an 'IncorrectState' exception is
+                 thrown.


    - remove_attribute
      Purpose:  removes an attribute.
      Format:   remove_attribute     (in string key);
      Inputs:   key:                  attribute to be removed
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - the attribute is not available anymore.
+     Perms:    Write
```

```
+     Throws:   NotImplemented
                DoesNotExist
-               ReadOnly
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - a vector attribute can also be removed with
                  this method
                - only some SAGA objects allow to remove
                  attributes.
                - a ReadOnly attribute cannot be removed - any
!                 attempt to do so throws a 'PermissionDenied'
                  exception.
                - if a non-existing attribute is removed, a
                  'DoesNotExist' exception is raised.
                - exceptions have the same semantics as defined
                  for the set_attribute() method description.


    - list_attributes
      Purpose:  Get the list of attribute keys.
      Format:   list_attributes      (out array<string>   keys);
      Inputs:   -
+     InOuts:   -
      Outputs:  keys:                 existing attribute keys
+     PreCond:  -
+     PostCond: -
+     Perms:    Query
+     Throws:   NotImplemented
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - exceptions have the same semantics as defined
                  for the get_attribute() method description.
                - if no attributes are defined for the object,
                  an empty list is returned.


    - find_attributes
      Purpose:  find matching attributes.
!     Format:   find_attributes      (in  array<string>   pattern,
-                                     in  array<string>   val_pattern,
```

```
                                        out array<string>   keys);
!     Inputs:   pattern:                search patterns
-               val_pattern:            value search pattern
+     InOuts:   -
      Outputs:  keys:                   matching attribute keys
+     PreCond:  -
+     PostCond: -
+     Perms:    Query
+     Throws:   NotImplemented
                BadParameter
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - the pattern must be formatted as described
                  earlier, otherwise a 'BadParameter' exception
                  is thrown.
                - exceptions have the same semantics as defined
                  for the get_attribute() method description.


    - attribute_is_readonly
!     Purpose:  check the attribute mode.
      Format:   attribute_is_readonly(in  string key,
                                        out bool   test);
      Inputs:   key:                    attribute key
+     InOuts:   -
!     Outputs:  test:                   bool indicating success
+     PreCond:  -
+     PostCond: -
+     Perms:    Query
+     Throws:   NotImplemented
                DoesNotExist
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - This method returns TRUE if the attribute
                  identified by the key exists, and can be read
                  by get_attribute() or get_vector attribute(),
                  but cannot be changed by set_attribute() and
                  set_vector_attribute().
                - exceptions have the same semantics as defined
                  for the get_attribute() method description.
```

```
    - attribute_is_writable
!     Purpose:  check the attribute mode.
      Format:   attribute_is_writable(in  string key,
                                      out bool   test);
      Inputs:   key:                  attribute key
+     InOuts:   -
!     Outputs:  test:                 bool indicating success
+     PreCond:  -
+     PostCond: -
+     Perms:    Query
+     Throws:   NotImplemented
                DoesNotExist
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - This method returns TRUE if the attribute
                  identified by the key exists, and can be
!                 changed by set_attribute() or
                  set_vector_attribute().
                - exceptions have the same semantics as defined
                  for the get_attribute() method description.


    - attribute_is_removable
!     Purpose:  check the attribute mode.
      Format:   attribute_is_removable (in  string key,
                                        out bool   test);
      Inputs:   key:                    attribute key
+     InOuts:   -
!     Outputs:  test:                   bool indicating success
+     PreCond:  -
+     PostCond: -
+     Perms:    Query
+     Throws:   NotImplemented
                DoesNotExist
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - This method returns TRUE if the attribute
                  identified by the key exists, and can be
```

```
                removed by remove_attribute().
              - exceptions have the same semantics as defined
                for the get_attribute() method description.


    - attribute_is_vector
      Purpose:  check the
      Format:   attribute_is_vector  (in  string key,
                                      out bool   test);
      Inputs:   key:                  attribute key
+     InOuts:   -
      Outputs:  test                  bool indicating if
                                      attribute is scalar
                                      (false) or vector (true)
+     PreCond:  -
+     PostCond: -
+     Perms:    Query
+     Throws:   NotImplemented
                DoesNotExist
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - This method returns TRUE if the attribute
                  identified by key is a vector attribute.
                - exceptions have the same semantics as defined
                  for the get_attribute() method description.
-
-
-   - attribute_equals
-     Purpose:  test for equality
-     Format:   attribute_equals     (in  string key,
-                                     in  string value,
-                                     out bool   test);
-     Inputs:   key:                  key   of attribute to test
-               value:                value to test
-     InOuts:   -
-     Outputs:  test                  bool indicating if
-                                     attribute has that value
-                                     (true) or not (false)
-     PreCond:  -
-     PostCond: -
-     Perms:    Query
-     Throws:   DoesNotExist
-               PermissionDenied
```

```
-               AuthorizationFailed
-               AuthenticationFailed
-               Timeout
-               NoSuccess
-     Notes:    - This method returns TRUE if the attribute
-                 identified by key has the value identified
-                 by value
-               - exceptions have the same semantics as defined
-                 for the get_attribute() method description.
```

3.8.4 Examples

Code Example

```
1  // c++ example:
2  job_definition d;
3
4  std::list <std::string> hosts;
5  hosts.push_back ("host_1");
6  hosts.push_back ("host_2");
7
8  // vector attributes
9  d.set_attribute ("ExecutionHosts", hosts);
10
11 // scalar attribute
12 d.set_attribute ("MemoryUsage", "1024");
13
14 ...
```

3.9 SAGA Monitoring Model

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

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

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

3.9.1 Specification

```
! package saga.monitoring
  {
!   // callbacks are used for asynchronous notification of
    // metric changes (events)
    interface callback
    {
      cb               (in  monitorable     mt,
                        in  metric          metric,
                        in  context         ctx,
                        out bool            keep);
    }


    // a metric represents an entity / value to be monitored.
    class metric : implements   saga::object
                   implements   saga::attributes
                // from object  saga::error_handler
    {
      CONSTRUCTOR        (in  string          name,
                          in  string          desc,
                          in  string          mode,
                          in  string          unit,
                          in  string          type,
```

```
                          in  string          value,
!                         out metric          obj);
!     DESTRUCTOR          (in  metric         obj);

      // callback handling
      add_callback        (in  callback       cb,
                           out int            cookie);
      remove_callback     (in  int            cookie);

      // actively signal an event
      fire                (void);


      // Attributes:
      //   name:  Name
!     //   desc:  name of the metric
      //   mode:  ReadOnly
      //   type:  String
      //   value: -
      //   notes: naming conventions as described below apply
      //
      //   name:  Description
!     //   desc:  description of the metric
      //   mode:  ReadOnly
      //   type:  String
      //
      //   name:  Mode
!     //   desc:  access mode of the metric
      //   mode:  ReadOnly
      //   type:  String
      //   value: 'ReadOnly', 'ReadWrite' or 'Final'
      //
      //   name:  Unit
!     //   desc:  unit of the metric
      //   mode:  ReadOnly
      //   type:  String
      //
      //   name:  Type
!     //   desc:  value type of the metric
      //   mode:  ReadOnly
      //   type:  String
      //   value: 'String', 'Int', 'Enum', 'Float', 'Bool',
      //          'Time' or 'Trigger'
      //
      //   name:  Value
!     //   desc:  value of the metric
```

```
    //   mode:  depending on the mode attribute above
    //   type:  String
    //   value: -
    //   notes: see description of value formating below
  }


  // SAGA objects which provide metrics and can thus be
  // monitored implement the monitorable interface
  interface monitorable
  {
    // introspection
    list_metrics       (out array<string>   names);
    get_metric         (in  string          name,
                        out metric          metric);

    // callback handling
    add_callback       (in  string          name,
                        in  callback        cb,
                        out int             cookie);
    remove_callback    (in  int             cookie);
  }


  // SAGA objects which can be steered by changing their
  // metrics implement the steerable interface
  interface steerable : implements monitorable
  {
    // metric handling
    add_metric         (in  metric          metric,
                        out bool            success);
    remove_metric      (in  string          name);
    fire_metric        (in  string          name);
  }
}
```

3.9.2 Specification Details

Interface `callback`

The callback interface is supposed to be implemented by custom, application level classes. Instances of these classes can then **be** passed to monitorable SAGA objects, in order to have their `cb` method invoked on changes of metrics **upon** these monitorables.

The callback classes can maintain state between initialization and successive **invocations**. The implementation MUST ensure that a callback is only called once at a time, so that no locking is neccessary for the end user.

If an invoked callback returns true, it stays registered and can be invoked again on the next metric change. If it returns false, it is not invoked again.

A callback can throw **an** `AuthorizationFailed` exception if the passed context (i.e. the remote party) is not deemed trustworthy. In this case, the callback is not removed. *The implementation MUST catch this exception, and interpret it as a decline of the operation which caused the callback.*

For example, if a `saga::stream_server` instance invokes a callback on a Client-Connect metric, and the `cb` method raises **an** `AuthorizationFailed` exception, the created client stream must be closed.

As another example, if a job instance invokes a callback on a MemoryUsage metric, and the `cb` method raises **an** `AuthorizationFailed` exception, the previous value of the memory usage metric MUST be restored, and the declined value MUST NOT influence the memory high water mark. **Essentially**, the exception indicates that the new metric value was not trustworthy.

Callbacks are passed (e.g. added to a metric) by reference. If a callback instance is used *multiple times with multiple metrics*, the application must use appropriate locking mechanisms.

```
    - cb
!     Purpose:  asynchronous handler for metric changes
      Format:   cb                 (in  monitorable mt,
                                    in  metric      metric,
                                    in  context     ctx,
                                    out bool        keep);
      Inputs:   mt:                 the saga monitorable object
!                                   which causes the callback
                                    invocation
                metric:             the metric causing the
                                    callback invocation
                ctx:                the context associated with
                                    the callback causing entity
+     InOuts:   -
      Outputs:  keep:               indicates if callback stays
                                    registered
+     PreCond:  - the passed context is authenticated.
+     PostCond: - if 'keep' is returned as true, the callback
!                 stays registered, and will be invoked again on
```

```
!                the next metric update.
!              - if 'keep' is returned as false, the callback
!                gets unregistered, and will not be invoked
!                again on metric updates, unless it gets
!                re-added by the user.
+     Perms:   -
+     Throws:  NotImplemented
               AuthorizationFailed
      Notes:   - 'metric' is the metric the callback is
                 invoked on - that means that this metric
                 recently changed.  Note that this change is
                 semantically defined by the metric, e.g. the
                 string of the 'value' attribute of the metric
                 might have the same value in two subsequent
                 invocations of the callback.
               - 'mt' is the monitorable object the metric
                 'metric' belonges to.
               - the context 'ctx' is the context which allows
                 the callback to authorize the metric change.
                 If the cb method decides not to authorize this
!                particular invocation, it MUST throw an
                 'AuthorizationFailed' exception.
               - if no context is available, a context of type
                 'Unknown' is passed, with no attributes
                 attached.  Note that this can also indicate
                 that a non-authenticated party connected.
               - a callback can be added to a metric multiple
!                times.  A 'false' return value (no keep) will
                 remove only one registration, and keep the
                 others.
               - a callback can be added to multiple metrics at
                 the same time.  A false return (no keep) will
                 only remove the registration on the metric the
                 callback was invoked on.
               - the application must ensure appropriate locking
!                of callback instances which are used with multiple
!                metrics.
               - a callback added to exactly one metric exactly
                 once is guaranteed to be active at most once at
!                any given time.  That implies that the SAGA
!                implementation MUST queue pending requests
!                until a callback invocation is finished.
```

Class `metric`

The fundamental object introduced in this package is a `metric`. A metric represents an observable *item*, which can be readable, or read/writable. The availability of a readable observable corresponds to monitoring; the availability of a writable observable corresponds to steering. A metric is `Final` when its values cannot change anymore, *ever* (i.e. progress is `100%`, job state is `Done` etc).

The approach is severely limited by the use of SAGA attributes for the description of a metric, as these are only defined in terms of string-typed keys and values. An extension of the attribute definition by typed values will greatly improve the usability of this package, but will also challenge its semantic simplicity.

The metric MUST provide access to following attributes (examples given):

```
name:      short human readable name.
             - ex:  file.copy.progress

desc:      extensive human readable description
             - ex:  "This metric gives the state of
                     an ongoing file transfer as
                     percent completed."

mode:      "ReadOnly", "ReadWrite" or "Final"
             - ex:  "ReadWrite"

unit:      Unit of values
             - ex:  "percent (%)"
             - ex:  "Unit"

type:      "String", "Int", "Enum", "Float", "Bool",
           "Time", "Trigger"
             - ex:   "Float"

value:     value of the metric
             - ex:   "20.5"
```

The name of the metric must be unique, as it is used in several methods to identify the metric of interest. The use of a dot-delimited name space for metrics as in the example above is encouraged, as it greatly benefits the interactive handling of metrics. The first element of the name space SHOULD be the SAGA

class the metric belongs to, the second element SHOULD be the operation the metric describes (if applicable, otherwise leave out), the third element SHOULD indicate the description of the metric (e.g. 'state' or 'progress' or 'temperature'). Illustrative examples for metric names are:

```
file.copy.progress
file.move.progress
file.size
job.state
drive.temperature  // a custom observable
```

The name, description, type and mode attributes are `ReadOnly` – so only unit and value can be changed by the application. All attributes are initialized in the metric constructor. The `mode`, `unit` and `value` attributes can be changed internally, i.e. by the SAGA implementation or lower layers. Such a change does cause the metric to *fire*. For example, a metric fires if its `mode` changes from `ReadWrite` to `Final`.

The `name` attribute MUST be interpreted case insensitive: An implementation MAY change that attribute to *all-lowercase* on metric creation.

If `fire()` is called on a metric, it returns immediately, but any callbacks registered on that metric are not invoked immediately. Instead, the remote entity which is represented by the metric gets invoked first, and only if it acknowledges the changes, the callbacks are invoked. A fire can thus fail in the sense that the remote entity declines the changes. It is good practice to have at least one callback registered on the metric before calling `fire()`, in order to confirm the operation.

The metric `type`s are the same as defined for attributes, and the metric `value`s are to be formatted as described for the respective attribute types. *The only exception is a metric of type `Trigger` which has no value at all – an attempt to access the value of that metric MUST result in a `DoesNotExist` exception.*

Metric definitions in the SAGA specification

The SAGA specification defines a number of metrics which MUST or CAN be supported, for various SAGA objects. An example of such a definition is (from the `saga::stream` object):

```
        class stream ...
        {
          ...

          // Metrics:
!         //   name:  stream.read
          //   desc:  fires if a stream gets readable
          //   mode:  ReadOnly
          //   unit:  1
          //   type:  Trigger
!         //   value: 1
          //
          //   ...
        }
```

These specifications are NORMATIVE, even if described as comments in the SIDL specification! The specified metrics MUST be supported by an implementation, unless noted otherwise in the mode description, as:

```
          //   mode:  ReadOnly, optional
          //   mode:  ReadWrite, optional
```

If a metric MUST be supported, but the SAGA implementation cannot provide that metric, any operation on that metric MUST throw a `NotImplemented` exception, and the resulting error message MUST state `"Metric <name> not not available in this implementation"`.

Implementations MAY add custom metrics, which SHOULD be documented similarly. However, metrics CAN also be added at runtime – that is, for example, required for computational steering of custom applications.

Metric Lifetime

A metric can *appear* and *go away* during the lifetime of an object (again, computational steering provides the obvious use case for this). Any operation on a metric which got removed (*dead metric*) MUST throw an `IncorrectState` exception, *with the exceptions described below*. Existing class instances of a dead metric MUST stay valid, and expose the same lifetime as any other ***live*** *metric*.

Attributes of a dead metric MUST be readable for the lifetime of the object. The `mode` attribute of such an instance MUST be changed to `Final` by the implementation. **Callbacks** cannot be registered to a `Final` metric, but can be unregistered. No other changes are allowed on a `Final` metric, neither by the user, nor by the SAGA implementation. *Allowed values for `mode` are `ReadOnly`, `ReadWrite`, and `Final`.*

Client Side Authorization

A metric can get fired from a remote party - in fact, that will be the default situation for both monitoring and steering. In order to allow for client side authorization, **callbacks** get a context as second parameter. That context contains information to be used to authorize the remote party which caused the metric to fire, and the callback to be invoked. Thus, authorization is only available via the callback mechanism. The context information passed to the callback are assumed to be authenticated by the implementation. If no context information **is** available, a context of type `'Unknown'` is passed, which has no attributes attached.

A callback can evaluate the passed context, and throw **an** *`AuthorizationFailed`* exception if the context (i.e. the remote party) is not deemed trustworthy. See callback description above.

```
    - CONSTRUCTOR
      Purpose:  create the object
      Format:   CONSTRUCTOR        (in  string   name
                                    in  string   desc,
                                    in  string   mode,
                                    in  string   unit,
                                    in  string   type,
                                    in  string   value,
                                    out metric   obj);
!     Inputs:   name:               name of the metric
!               desc:               description of the metric
!               mode:               mode of the metric
!               unit:               unit of the metric value
!               type:               type of the metric
!               value:              initial value of the metric
+     InOuts:   -
      Outputs:  obj:                the newly created object
+     PreCond:  -
+     PostCond: - callbacks can be registered on the metric.
+     Perms:    -
```

```
      Throws:   NotImplemented
                BadParameter
                Timeout
                NoSuccess
      Notes:    - a metric is not attached to a session, but
                  can be used in different sessions.
                - the string arguments given are used to
!                 initialize the attributes of the metric.
-                 , which
-                 are subsequently ReadOnly (see description
-                 above).
                - the constructor ensures that metrics are
                  always initialized completely.  All changes to
                  attributes later will always result in an
                  equally valid metric.
                - incorrectly formatted 'value' parameter,
                  invalid 'mode' and 'type' parameter, and empty
                  required parameter (all but 'unit') will cause
                  a 'BadParameter' exception.
                - a 'Timeout' or 'NoSuccess' exception indicates
                  that the backend could not create that specific
                  metric.


    - DESTRUCTOR
      Purpose:  destroy the object
      Format:   DESTRUCTOR         (in  metric obj)
      Inputs:   obj:                the object to destroy
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - all callbacks registered on the metric are
!                 unregistered.
+     Perms:    -
      Throws:   -
+     Notes:    - if a callback is active at the time of
!                 destruction, the destructor MUST block until
!                 that callback returns.  The callback is not
!                 activated anew during or after that block.


    // manage callbacks on the metric
    - add_callback
!     Purpose:  add asynchronous notifier callback to watch
                metric changes
      Format:   add_callback       (in  callback cb,
```

```
                                    out int      cookie);
      Inputs:   cb:                 callback class instance
+     InOuts:   -
      Outputs:  cookie:             handle for this callback,
                                    to be used for removal
+     PreCond:  - the metric is not 'Final'.
+     PostCond: - the callback is invoked on metric changes.
+     Perms:    Read
      Throws:   NotImplemented
-               WriteOnly
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - 'IncorrectState' is thrown if the metric is
                  'Final'.
                - the 'callback' method on cb will be invoked on
                  any change of the metric (not only when its
                  value changes)
                - if the 'callback' method returns true, the
                  callback is kept registered; if it returns
                  false, the callback is called, and is
                  un-registered after completion.  If the
                  callback throws an exception, it stays
                  registered.
                - the cb is passed by reference.
                - the returned cookie uniquely identifies the
                  callback, and can be used to remove it.
!               - A 'Timeout' or 'NoSuccess' exception is thrown
!                 if the implementation cannot in voke the
!                 callback on metric changes.
                - a backend MAY limit the ability to add
                  callbacks - the method may hence cause an
                  'AuthenticationFailed', 'AuthorizationFailed'
                  or 'PermissionDenied' exception to be thrown.


    - remove_callback
      Purpose:  remove a callback from a metric
-               changes
      Format:   remove_callback    (in  int      cookie);
      Inputs:   cookie:             handle identifying the cb to
                                    be removed
+     InOuts:   -
```

```
      Outputs:  -
+     PreCond:  - the callback identified by 'cookie' is
+                 registered for that metric.
+     PostCond: - the callback identified by 'cookie' is not
+                 active, nor invoked ever again.
+     Perms:    Read
      Throws:   NotImplemented
                BadParameter
-               WriteOnly
+               PermissionDenied
+               AuthorizationFailed
+               AuthenticationFailed
                Timeout
                NoSuccess
+     Notes:    - if a callback is active at the time of
+                 removal, the call MUST block until
+                 that callback returns.  The callback is not
+                 activated anew during or after that block.
                - if the callback was removed earlier, or
                  was unregistered by returning false, this call
                  does nothing.
                - the removal only affects the cb identified
                  by 'cookie', even if the same callback was
                  registered multiple times.
                - if the cookie was not created by adding a
                  callback to this object instance, a
                  'BadParameter' is thrown.
                - a 'Timeout' or 'NoSuccess' exception is thrown
                  if the backend cannot guarantee that the
                  callback gets successfully removed.
                - note that the backend MUST allow the removal of
                  the callback, if it did allow its addition -
                  hence, no authentication, autorization or
!                 permission faults are tom be expected.


    - fire
      Purpose:  push a new metric value to the backend
      Format:   fire               (void);
      Inputs:   -
+     InOuts:   -
      Outputs:  -
+     PreCond:  - the metric is not 'Final'.
+               - the metric is 'ReadWrite'
+     PostCond: - callbacks registered on the metric are
+                 invoked.
```

```
+     Perms:    Write
      Throws:   NotImplemented
-               ReadOnly
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - 'IncorrectState' is thrown if the metric is
                  'Final'.
                - 'PermissionDenied' is thrown if the metric is
!                 not 'ReadWrite' -- That also holds for a once
                  writable metric which was flagged 'Final'.
!                 To catch race conditions on this exceptions,
!                 the application should try/catch the fire().
                - it is not necessary to change the value of a
                  metric in order to fire it.
                - 'set_attribute ("value", "...") on a metric
                  does NOT imply a fire.  Hence the value can be
                  changed multiple times, but unless fire() is
!                 explicitly called, no consumer will notice.
                - if the application invoking fire() has
                  callbacks registered on the metric, these
!                 callbacks are invoked.
                - 'AuthenticationFailed', 'AuthorizationFailed'
                  or 'PermissionDenied' may get thrown if the
                  curent session is not allowed to fire this
                  metric.
                - a 'Timeout' or 'NoSuccess' exception signals
                  that the implementation could not communicate
                  the new metric state to the backend.
```

Interface `monitorable`

The monitorable interface is implemented by those SAGA objects which can be monitored, i.e. which have one or more associated metrics. The interface allows introspection of these metrics, and allows to add callbacks to these metrics which get called if these metrics change.

Several methods of this interface reflect similar methods on the metric class – the additional string argument `name` identifies the metric these methods act upon. The semantics of these calls are identical to the specification above.

```
    // introspection
    - list_metrics
      Purpose:  list all metrics associated with the object
      Format:   list_metrics       (out array<string>   names);
      Inputs:   -
+     InOuts:   -
      Outputs:  names:              array of names identifying
                                    the metrics associated with
                                    the object instance
+     PreCond:  -
+     PostCond: -
+     Perms:    Query
      Throws:   NotImplemented
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - several SAGA objects are required to expose
                  certain metrics (e.g. 'task.state').  However,
                  in general that assumption cannot be made, as
                  implementations might be unable to provide
                  metrics.  In particular, listed metrics might
!                 actually be unavailable.
                - no order is implied on the returned array
                - the returned array is guaranteed to have no
                  double entries (names are unique)
                - an 'AuthenticationFailed',
                  'AuthorizationFailed' or 'PermissionDenied'
                  exception indicates that the current session
                  is not allowed to list the available metrics.
                - a 'Timeout' or 'NoSuccess' exception indicates
                  that the backend was not able to list the
                  available metrics.


    - get_metric
      Purpose:  returns a metric instance, identified by name
      Format:   get_metric         (in  string  name,
                                    out metric  metric);
      Inputs:   name:               name of the metric to be
                                    returned
+     InOuts:   -
      Outputs:  metric:             metric instance identified
```

```
                                      by name
+     PreCond:  -
+     PostCond: -
+     Perms:    Query
      Throws:   NotImplemented
                DoesNotExist
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - multiple calls of this method with the same
                  value for name return multiple identical
                  instances (copies) of the metric.
                - a 'DoesNotExist' exception indicates that the
                  backend does not know the metric with the
                  given name.
                - an 'AuthenticationFailed',
                  'AuthorizationFailed' or 'PermissionDenied'
                  exception indicates that the current session
                  is not allowed to obtain the named metric.
                - a 'Timeout' or 'NoSuccess' exception indicates
                  that the backend was not able to return the
                  named metric.


    // callback handling
    - add_callback
      Purpose:  add a callback to the specified metric
      Format:   add_callback       (in  string      name,
                                    in  callback    cb,
                                    out int         cookie);
!     Inputs:   name:               identifies the metric to
                                    which cb
                                    is to be added
-               cb:                 reference of callback class
+               cb:                 reference to callback class
                                    instance to be registered
+     InOuts:   -
      Outputs:  cookie:             handle for callback removal
+     PreCond:  -
+     PostCond: - the callback is registered on the metric.
+     Perms:    Read on the metric.
      Throws:   NotImplemented
                DoesNotExist
                IncorrectState
```

```
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
                NoSuccess
      Notes:    - notes to the add_callback method of the metric
                  class apply.


    - remove_callback
      Purpose:  remove a callback from the specified metric
      Format:   remove_callback    (in  string  name,
                                    in  int     cookie);
      Inputs:   name:               identifies the metric for
!                                   which cb is to be removed
                cookie:             identifies the cb to be
                                    removed
+     InOuts:   -
+     Outputs:  -
+     PreCond:  - the callback was registered on the metric.
+     PostCond: -
+     Perms:    Read on the metric.
      Throws:   NotImplemented
                BadParameter
                DoesNotExist
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - notes to the remove_callback method of the
                  metric class apply
```

Interface `steerable`

The steerable interface is implemented by saga objects which can be steered, i.e. which have writable metrics, and which might allow to add new metrics. Steerable objects *must* also implement the monitorable interface.

The method `add_metric()` allows to implement steerable applications. In particular, the `saga::self` object is steerable, and allows to add metrics (see description of `saga::self` in the specification of the SAGA job management).

```
    // metric handling
    - add_metric
      Purpose:  add a metric instance to the application
                instance
      Format:   add_metric         (in  metric  metric,
                                    out bool    success);
      Inputs:   metric:             metric to be added
+     InOuts:   -
      Outputs:  success:            indicates success
+     PreCond:  -
+     PostCond: - the metric can be accessed from this
+                 application, and possibly from other
+                 applications.
+     Perms:    Write
      Throws:   NotImplemented
                AlreadyExists
-               ReadOnly
+               IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - a metric is uniquely identified by its name
                  attribute - no two metrics with the same name
                  can be added.
                - any callbacks already registered on the metric
!                 stay registered (the state of metric is not
                  changed)
!               - an object being steerable does not guarantee
                  that a metric can in fact be added -- the
                  returned boolean indicates if that particular
                  metric could be added.
                - an 'AuthenticationFailed',
                  'AuthorizationFailed' or 'PermissionDenied'
                  exception indicates that the current session
                  is not allowed to add metrics to the
                  steerable.
                - a 'Timeout' or 'NoSuccess' exception indicates
                  that the backend was not able to add the
                  metric.
                - if a metric with the same name is already
                  known for the object, an 'AlreadyExists'
                  exception is thrown.
```

```
                - if the steerable instance does not support the
                  addition of new metrics, i.e. if only the
                  default metrics can be steered, an
!                 'IncorrectState' exception is thrown.


    - remove_metric
      Purpose:  remove a metric instance
      Format:   remove_metric      (in  string  name);
      Inputs:   name:               identifies the metric to be
                                    removed
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - all callbacks registered on that metric are
+                 unregistered.
+               - the metric is not available anymore.
+     Perms:    Write
      Throws:   NotImplemented
                DoesNotExist
-               ReadOnly
+               IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - only previously added metrics can be removed;
!                 default metrics (saga defined or implementation
!                 specific) cannot be removed; attempts to do so
                  raise a BadParameter exception.
                - an 'AuthenticationFailed',
                  'AuthorizationFailed' or 'PermissionDenied'
                  exception indicates that the current session
                  is not allowed to remove the metrics from the
                  steerable.
                - a 'Timeout' or 'NoSuccess' exception indicates
                  that the backend was not able to remove the
                  metric.
                - if a metric with that name is not known for
                  the object, a 'DoesNotExist' exception is
                  thrown.
                - if a steerable instance does not support the
                  removal of some metric, e.g. if a metric
                  needs to be always present, an
!                 'IncorrectState' exception is thrown.
```

```
                  For example, the 'state' metric on a steerable
                  job cannot be removed.


    - fire_metric
      Purpose:  push a new metric value to the backend
      Format:   fire_metric        (int  string  name);
      Inputs:   name:               identifies the metric to be
                                    fired
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: -
+     Perms:    Write
      Throws:   NotImplemented
                DoesNotExist
-               ReadOnly
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - notes to the fire method of the metric
                  class apply
                - fire can be called for metrics which have been
                  added with add_metric(), and for predefined
                  metrics
                - an 'AuthenticationFailed',
                  'AuthorizationFailed' or 'PermissionDenied'
                  exception indicates that the current session
                  is not allowed to fire the metric.
                - a 'Timeout' or 'NoSuccess' exception indicates
                  that the backend was not able to fire the
                  metric.
                - if a metric with that name is not known for
                  the object, a 'DoesNotExist' exception
                  is thrown.
!               - an attempt to fire a metric which is
!                 'ReadOnly' results in an 'IncorrectState'
                  exception.
                - an attempt to fire a 'Final' metric results in
!                 an 'IncorrectState' exception.
```

3.9.3 Examples

Code Example

```
callback example: trace all job state changes:
----------------------------------------------

  // c++ example
  // callback definition
  class trace_cb : public saga::callback
  {
    public:
      bool cb (saga::monitorable mt,
               saga::metric      m,
               saga::context     c)
      {
        std::cout << "metric " << m.get_attribute ("name")
                  << " fired." << std::endl;
        return true; // stay registered
      }
  }

  // the application
  int main ()
  {
    ...

    // if the callback defined above is added to all known
    // metrics of all saga objects, a continous trace of state
    // changes of these saga objects will be written to stdout
    trace_cb cb;

    saga::job j = ...

    j.add_callback ("state", cb);

    ...
  }


monitoring example: monitor a write task
----------------------------------------

  // c++ example for task state monitoring
  class write_metric_cb : public saga::callback
  {
    public:
      bool cb (saga::monitorable mt,
               saga::metric      m,
               saga::context     c)
```

```
47        {
48          saga::task t = saga::task (mt);
49
50          std::cout << "bytes written: "
51                    << m.get_attribute ("value")
52                    << std::endl;
53          std::cout << "task state:   "
54                    << t.get_state ()
55                    << std::endl;
56
57          return true; // keep callback registered
58        }
59    };
60
61    int main (int argc, char** argv)
62    {
63      ssize_t      len = 0;
64      saga::buffer buf ("Hello SAGA\n");
65      saga::url    url (argv[1]);
66
67      saga::file   f (url);
68      saga::task   t = f.write <saga::task::Async> (buf, &len);
69
70      // assume that a file write task has a 'progress' metric
71      // indicating the number of bytes already written.  In
72      // general, the list of metric names has to be searched
73      // for an interesting metric, unless it is a default
74      // metric as specified in the SAGA spec.
75
76      // create and add the callback instance
77      write_metric_callback cb;
78      t.add_callback ("file.write.progress", cb);
79
80      // wait until task is done, and give cb chance to get
81      // called a couple of times
82      t.wait ();
83    }
84
85
86  steering example: steer a remote job
87  ------------------------------------
88
89    // c++ example
90    class observer_cb : public saga::metric::callback
91    {
92      public:
93        bool cb (saga::monitorable mt,
94                 saga::metric      m,
95                 saga::context     c)
96        {
```

```
97            std::cout << "the new value is"
98                      << atoi ( m.get_attribute ("value") )
99                      << std::endl;
100
101           return true; // keep callback registered
102       }
103   };
104
105   // the steering application
106   int main (int argc, char** argv)
107   {
108     saga::job_service js;
109
110     saga::job j = js.run ("remote.host.net",
111                           "my_remote_application");
112
113     // Assume that job has a 'param_1' metric representing
114     // an integer parameter for the remote application.
115     // In general, one has to list the metrics available on
116     // job, with list_metric, and search for an interesting
117     // metric.  However, we assume here that we know that
118     // metric exists.  So we get that metric, and add an
119     // observer callback to it - that causes the asynchronous
120     // printout of any changes to the value of that metric.
121
122     // then we get the metric for active steering
123     saga::metric m = j.get_metric ("param_1");
124
125     observer_cb cb;
126     m.add_callback (cb);
127
128     for ( int i = 0; i < 10; i++ )
129     {
130       // if param_1 is ReadOnly, set_value() would throw
131       // 'ReadOnly' - it would not be usable for
132       // steering then.
133       m.set_attribute ("value", std::string (i));
134
135       // push the pending change out to the receiver
136       m.fire ();
137
138       // callback should get called NOW + 2*latency
139       // That means fire REQUESTS the value change, but only
140       // the remote job can CHANGE the value - that change
141       // needs then reporting back to us.
142
143       // give steered application some time to react
144       sleep (1);
145     }
146   }
```

```
147
148
149
150  steering example: BE a steerable job
151  ------------------------------------
152
153    // c++ example
154    //
155    // the example shows a job which
156    //  - creates a metric to expose a Float steerable
157    //    parameter
158    //  - on each change of that parameter computes a
159    //    new isosurface
160    //
161    // callback - on any change of the metric value, e.g. due to
162    // steering from a remote GUI application, a new iso surface
163    // is computed
164    class my_cb : public saga::callback
165    {
166      public:
167        // the callback gets called on any steering events, i.e.
168        // if some other application steeres 'me'.
169        bool cb (saga::monitorable mt,
170                 saga::metric      m,
171                 saga::context     c)
172        {
173          // get the new iso-value
174          float iso = atof (m.get_attribute ("value"));
175
176          // compute an isosurface with that iso-value
177          compute_iso (iso);
178
179          // keep this callback alive, and get called again on
180          // the next metric event.
181           return true;
182        }
183     }
184
185    int main ()
186    {
187      // create a metric for the iso-value of an isosurfacer
188      saga::metric m ("application.isosurfacer.isovalue",
189                      "iso-value of the isosurfacer",
190                      "ReadWrite",   // is steerable
191                      "",            // no unit
192                      "Float",       // data type
193                      "1.0");        // initial value
194
195      // add the callback which reacts on changes of the
196      // metric's value (returned cookie is ignored)
```

```
197       my_cb cb;
198       m.add_callback (cb);
199
200       // get job handle for myself
201       saga::self self;
202
203       // add metric to myself
204       self.add_metric (m);
205
206       /*
207       // the callback could also have been added with:
208       self.add_callback ("application.isosurfacer.isovalue", cb);
209       */
210
211       // now others can 'see' the metric, e.g. via
212       // job.list_metrics ();
213
214       // compute isosurfaces for the next 10 minutes -
215       // the real work is done in the callback, on incoming
216       // requests (i.e. steering events).
217       sleep (600);
218
219       // on object (self) destruction, metrics and callback
220       // objects are destroyed as well
221       return (0);
222     }
223
224
225
226   monitoring example: callback for stream connects
227   -----------------------------------------------
228
229     // c++ example
230     //
231     // callback class which accepts an incoming client
232     // connection, and then un-registeres itself.  So, it
233     // accepts exactly one client, and needs to be re-registered
234     // to accept another client.
235     class my_cb : public saga::callback
236     {
237       privat:
238         // we keep a stream server and a single client stream
239         saga::stream_server ss_;
240         saga::stream        s_;
241
242
243       public:
244         // constructor initializes these (note that the
245         // client stream should not be connected at this
246         // point)
```

```
247         my_cb (saga::stream_server ss,
248                saga::stream        s )
249         {
250           ss_ = ss;
251           s_  = s;
252         }
253
254
255         // the callback gets called on any incoming client
256         // connection
257         bool cb (saga::monitorable mt,
258                  saga::metric      m,
259                  saga::context     c)
260         {
261           // the stream server got an event triggered, and
262           // should be able to create a client socket now.
263           s_ = ss_.wait ();
264
265           if ( s_.state == saga::stream::Open )
266           {
267             // have a client stream, we are done
268             // don't call this cb again!
269             return (true);
270           }
271
272           // no valid client stream obtained: keep this
273           // callback alive, and get called again on the
274           // next event on ss_
275           return true;
276         }
277       }
278
279       int main ()
280       {
281         // create a stream server, and an un-connected
282         // stream
283         saga::stream_server ss;
284         saga::stream        s;
285
286         // give both to our callback class, and register that
287         // callback with the 'client_connect' metric of the
288         // server.  That causes the callback to be invoked on
289         // every change of that metric, i.e. on every event
290         // that changes that metric, i.e. on every client
291         // connect attempt.
292         my_cb cb (ss, s);
293         ss.add_callback ("client_connect", cb);
294
295         // now we serve incoming clients forever
296         while ( true )
```

```
297       {
298         // check if a new client is connected
299         // the stream state would then be Open
300         if ( s.state == saga::stream::Open )
301         {
302           // a client got conncted!
303           // handle open socket
304           saga::buffer buf ("You say hello, "
305                             "I say good bye!\r\n", 33);
306           s.write (buf);
307
308           // and close stream
309           s.close ();
310
311           // the stream is not Open anymore.  We re-add the
312           // callback, and hence wait for the next client
313           // to connect.
314           ss.add_callback ("client_connect", cb);
315         }
316         else
317         {
318           // no client yet, idle, or do something useful
319           sleep (1);
320         }
321       }
322
323       // we should never get here
324       return (-1);
325     }
```

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.

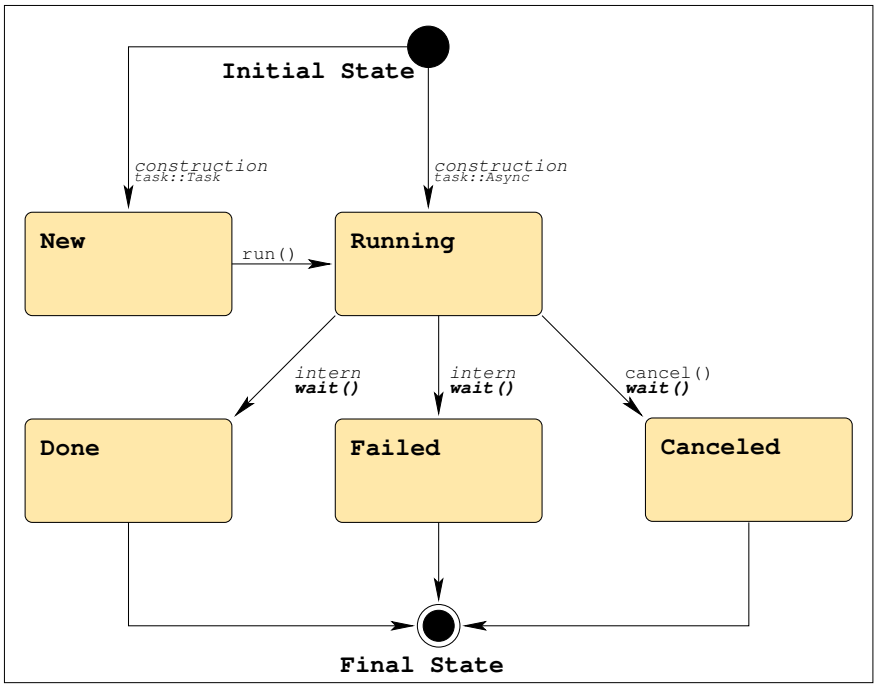

Figure 3: The SAGA task state model (See figure 1 for a *legend*).

In order to understand the SAGA task model it is *not* sufficient to read the specification of the `saga::task` and `saga::task_container` classes below, but it is also imperative to understand how task instances get created. This is actually not covered in the SIDL specification sections in this document, but documented in prose below, with references to Figure 3. Note that the task state model is closely modeled after the BES state model [12], which is in particular relevant to the (similar) job state model as described in Section 4.1.

Tasks versus Jobs

In SAGA, tasks should not be confused with jobs! Jobs represent remotely running applications/executables, which are ussually managed by a job manager. Tasks on the other hand represent asynchronous operations. Thus, any asynchronous method call in SAGA results in a task.

Tasks and jobs have, however, several commonalities, the most important one is state: both can be newly created (in `New` state), can be currently making progress (in `Running` state), or can be finished in some way (in `Done`, `Failed` or `Canceled` state). Additionally, jobs can be suspended and resumed (they have a `Suspended` state).

Mostly for this reason, and to simplify the management of both tasks and jobs in SAGA, the `saga::job` class inherits the `saga::task` class.

Tasks versus Threads

Tasks and threads are another potential pair to confuse: in many APIs and programming languages, tasks and asynchronous operations are implemented by threading. In SAGA, however, 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.

For example, a SAGA task to copy a remote file could be implemented by using the Globus Reliable File Transfer Service (RFT, [1]): the asynchronous method invocation in SAGA would then start the remote operation on the RFT service. All management of the operation progress is in the service - no threading at all is required on the application side. Even more: the application could finish, and after restart could reconnect to the RFT service, and recreate the task, as the complete state is still available on the RFT service - that is basically impossible with threads. Well, it is also not possible in SAGA right now, but for very different reasons, and it is expected that future versions and extensions of SAGA add this and other options to the notion of tasks.

*Implementors of SAGA are warned **not** to rely solely on threading while implementing `saga::task`, but to exploit middleware support for server side asynchronous operations wherever possible.*

Task Model Description

The SAGA task model operates as follows:

- A SAGA object is said to *implement the SAGA task model* if, (a) it inherits the `saga::async` interface, and (b) all methods on that object are implemented in three different versions, which are called *synchronous*, *asynchronous*, and *task* version.
- The *synchronous* version of a SAGA call **corresponds** to the normal method call specified in the SAGA specification. The first `out` parameter specified (if any) is used as return value.
- The *asynchronous* version of ~~SAGA a~~ *a SAGA* call has ~~a different~~ *the same* signature, *but* **returns** a `saga::task` instance. That returned task is in `Running` state and represents the asynchronous operation: it can be queried for state, and can be canceled.
- The *task* version of the SAGA call is very similar to the asynchronous version; the only difference is that the returned task instance is in the `New` state, and must be `run()` to get into the `Running` state.
- For symmetry, a language binding MAY add a second flavour of the synchronous call, which has the same signature as **the** asynchronous and task version, but the returned task is in a final state (i.e., `run()` *and* `wait()` have been called on that task before returning). [2]
- *The first* `out` *parameter, which is the return value in the synchronous method version, is, in the task and asynchronous version, accessed by calling* `task.get_result <return_type> (void);`*, which is a templetized member method. That call implies a call to* `wait()`*. For language bindings where templetized member functions are not available, a language specific mechanism MUST be found, which MAY use type casting.*
- *Other* `out` and all `inout` parameter for asynchronous operations *are passed by reference to the initial function call, and* MUST NOT be accessed before the corresponding task enters *the* `Done` state. In all other states, no assumption can be made about the contents of these parameters. *They are guaranteed to not be accessed or changed by the implementation when the task enters any final state.*
- `in` parameters are passed by value, and are assumed to be constant. They can be accessed and changed again as soon as the task instance is created.
- *The original object instance, from which the task was created, can be retrieved from a task by calling* `get_object <object_type> (void);`*, again a templetized member method, on the task. The same comments as above apply to that templetized method.*

[2] Note that state transitions for this type of method call are not shown in the state diagram – the diagram would essentially have 'Done' as **an** inital and final state.

Asynchronous Object Construction

The task model as described above focuses on asynchronous invocation of object methods. It does not explicitly cover asynchronous object construction or destruction though. That is important, however, as many constructors, such as for example for `saga::file`, imply a remote operation during construction or destruction (here `open()/close()`).

How asynchronous constructors and destructors are provided is up to the specific language bindings. Procedural bindings, such as expected for C, SHOULD integrate asynchronous versions for the respective method calls to keep these mechanisms in sync with the task model presented above. Object oriented language bindings MAY either introduce an asynchronous factory pattern, or introduce delayed construction/destruction by explicitly using asynchronous `init()` and `close()`, or MAY introduce some other mechanism which most natively allows to asynchronously create SAGA objects.

Tasks and Error Handling

Errors arising from synchronous method invocations on SAGA objects are, in general, flagged by exceptions, and can be inspected using the `error_handler` interface that all SAGA objects implement. For asynchronous operations, this mechanism would break, as the `error_handler` interface allows *in general* only inspection of the *last* method call – but the order of execution is undefined for asynchronous operations. Additionally, exceptions from asynchronous operations would be difficult to catch, as they would presumably be thrown outside of any exception protection block.

For **this** reason, errors on asynchronous operations (i.e. tasks) are handled as follows:

Error Handler: The `saga::task` class implements the `saga::error_handler` interface, which allows inspection of an error thrown by an asynchronous operation. Errors MUST NOT be reported unless the task enters a final state.

Exceptions: The task instance MUST catch all SAGA exceptions and, if possible, all other exceptions thrown by the asynchronous operation. If an exception is caught by the task instance, the task state MUST be changed to `Failed` immediately. Such exceptions are to be re-thrown by the task when the `rethrow()` method is called.

This specification assumes that tasks are, in general, created and maintained in the API implementation, and not in the backend. However, for those cases

where task states are maintained in the middleware backend, several methods on `tasks` and `task_containers` MAY throw a `Timeout` or `NoSuccess` exception, if that backend is not available – *these exceptions can be directly delivered to the application*. It is, however, not allowed to throw an `AuthorizationFailed`, `AuthenticationFailed` or `PermissionDenied` exception, as this **specification** assumes that the creator of the task can always inspect and control that task – *these exceptions MUST be caught, and MUST be made available via `rethrow()`*. Later versions of this API MAY change that, for example when they introduce persistent tasks which can survive the lifetime of a SAGA application.

3.10.1 Example Rendering in C++

Below is an example of how the SAGA task model might be rendered in C++ (this example is not normative). Note that template-tags are used here to distinguish the three task-returning method calls.

Code Example

```
1  // c++ example
2
3  // SAGA specfication:
4  //  read        (inout array<byte>             buffer,
5  //               in    int                     len_in = -1,
6  //               out   int                     len_out);
7
8  // create a saga file
9  saga::file f (url);
10
11 // synchronous version
12 ssize_t len_out = f.read (size_t   len_in,
13                            char   * buffer);
14
15
16 // alternative synchronous version
17 saga::task t1   = f.read <saga::task::Sync>
18                           (size_t    len_in,
19                            char    * buffer);
20
21 // asynchronous version
22 saga::task t2   = f.read <saga::task::ASync>
23                           (size_t    len_in,
24                            char    * buffer);
25
26 // asynchronous version
27 // task version
28 saga::task t3   = f.read <saga::task::Task>
29                           (size_t    len_in,
30                            char    * buffer);
```

```
31
32  // t1 is in Done or Failed state
33  // t2 is in Running state
34  // t3 is in New state
35
36  // get results
37  ssize_t len_out_1 = t1.get_result <ssize_t> ();
38  ssize_t len_out_2 = t2.get_result <ssize_t> ();
39  ssize_t len_out_3 = t3.get_result <ssize_t> ();
40
41  // all rasks are in a final state now,
42  // as get_result() implies a wait().
43
44  // obtain the original file object, three
45  // times the same actually
46  saga::file f1 = t1.get_object <saga::file> ();
47  saga::file f2 = t2.get_object <saga::file> ();
48  saga::file f3 = t3.get_object <saga::file> ();
```

A C language binding of this package might choose to use flags to distinguish *between* these calls; equivalently the C binding might use different method names, for it is up to the language bindings to define the mechanism that is native – or as close as possible – to the language to distinguish these calls.

Note that a SAGA task represents an asynchronous version of a SAGA API method call, and as such it may, or may not have a one-to-one correspondence to an external process, thread, or operation handle.

In general, care should be exercised not to confuse tasks and jobs, as they represent different paradigms: a SAGA job explicitly and always represents an externally running executable, performing any kind of work and as such IS-A task; whereas the internal representation of a SAGA task is very much up to the implementation, and a task is not always a job, i.e. is not always performed by running an executable.

It should also be noted that the task state model (see Fig. 3) and the job state model (see Fig. 4) are very similar, in that the task states represent a subset of the job state model (as can be expected, for a job IS-A task).

For additional notes on resource management and task lifetime, see the introduction Section 2.5.3 of this document.

3.10.2 Specification

```
package saga.task
{
  enum state
```

```
    {
      New       =  1,
      Running   =  2,
      Done      =  3,
      Canceled  =  4,
      Failed    =  5
    }


    enum wait_mode
    {
      All       =  0,
      Any       =  1
    }


    interface async
    {
      // this interface is empty on purpose, and is used only
      // for tagging of SAGA classes which implement the SAGA
      // task model.
    }


    class task : implements   saga::object
                 implements   saga::monitorable
              // from object  saga::error_handler
    {
!     // no constructor
      DESCTRUCTOR       (in  task            obj);

+     // state management
      run               (void);
      cancel            (in  float           timeout =  0.0);
      wait              (in  float           timeout = -1.0,
                         out boolean         finished);

+     // inspection
      get_state         (out state           state);
+     get_result <type> (out type            result);
+     get_object <type> (out type            object);

+     // error handling
      rethrow           (void);

      // Metric:
```

```
      //   name:  task.state
!     //   desc:  fires on task state change, and
      //          has the literal value of the task
      //          state enum.
      //   mode:  ReadOnly
      //   unit:  1
!     //   type:  Enum
      //   value: 0
    }


    class task_container : implements   saga::object
                           implements   saga::monitorable
                         // from object  saga::error_handler
    {
      CONSCTRUCTOR      (out task_container  obj);
      DESCTRUCTOR       (in  task_container  obj);

+     // task management
      add               (in  task            task,
                          out int            cookie);
      remove            (in  int             cookie,
+                         out task           task);

+     // state management
      run               (void);
      cancel            (in  float           timeout =  0.0);
!     wait              (in  wait_mode       mode    =  All,
!                         in  float          timeout = -1.0,
                          out task           finished);

+     // inspection
+     size              (out int             n);
      list_tasks        (out array<int>      cookies);
+     get_task          (in  int             cookie,
+                         out task           t);
      get_tasks         (out array<task>     tasks);
      get_states        (out array<state>    states);

      // Metric:
!     //   name:  task_container.state
      //   desc:  fires on state changes of any task in
      //          container, and has the value of that
      //          task's cookie.
      //   mode:  ReadOnly
      //   unit:  1
```

```
      //   type:  Int
      //   value: -
    }
  }
```

3.10.3 Specification Details

Enum `state`

A task can be in one of several possible states *(see Fig. 3)*:

New: The task has been created but not yet started. Tasks start in this state, it is initial.

`New`

This state identifies a newly **constructed** task instance which has not yet run. This state corresponds to the BES state 'Pending'. This state is initial.

`Running`

The `run()` method has been invoked on the task, either explicitly or implicitly. This state corresponds to the BES state 'Running'. This state is initial.

`Done`

The synchronous or asynchronous operation has finished successfully. It corresponds to the BES state 'Finished'. This state is final.

`Canceled`

The asynchronous operation has been canceled, i.e. `cancel()` has been called on the task instance. It corresponds to the BES state 'Canceled'. This state is final.

`Failed`

The **synchronous** or asynchronous operation has finished unsuccessfully. It corresponds to the BES state 'Failed'. This state is final.

Enum `wait_mode`

The `wait_mode` enum specifies the condition on which a `wait()` operation on a `saga::task_container` returns:

`All`

wait() returns if all tasks in the container reached a final state.

`Any`

wait() returns if one or more tasks in the container reached a final state.

Class `task`

Objects of this class represent asynchronous API calls. They are only created by invoking a method on a **SAGA** object which returns a task object (with `saga::task::ASync` or `saga::task::Task`). But as `saga::job` instances inherit from the task class, *tasks jobs* are also effectively created as *jobs tasks*.

If a task gets created, it will share the state of the object it was created from. For more information on state sharing, see *Section 2.5.3*).

Note that no CONSTRUCTOR is available, as tasks are only created through asynchronous method calls.

```
    - DESTRUCTOR
      Purpose:  destroy the object
      Format:   DESTRUCTOR           (in  task obj)
      Inputs:   obj:                  the object to destroy
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - state is no longer shared with the object
+               - the task was created from.
!               - the task instance is 'Canceled' prior to
+               - resource deallocation.
+     Perms:    -
      Throws:   -
      Notes:    - if the instance was not in a final state
                - before, the destructor performs a cancel()
                - on the instance, and all notes to cancel()
                - apply.


+   State Management
+   ----------------
+
    - run
      Purpose:  Start the asynchronous operation.
```

```
      Format:   run (void);
      Inputs:   -
+     InOuts:   -
      Outputs:  -
+     PreCond:  - task is in 'New' state.
+     PostCond: - task is in 'Running' state.
+     Perms:    - appropriate permissions for the method
+                 represented by the task
      Throws:   NotImplemented
                IncorrectState
                Timeout
                NoSuccess
      Notes:    - run can only be called on a task in 'New'
                  state.  All other states will cause the
                  'IncorrectState' exception to be thrown.
                - a 'Timeout' or 'NoSuccess' exception indicates
                  that the backend was not able to start the
                  task.


    - wait
      Purpose:  Wait for the task to finish.
      Format:   wait                 (in  float   timeout,
                                      out boolean done);
      Inputs:   timeout:              seconds to wait
+     InOuts:   -
      Outputs:  done:                 indicating if the task
                                      is done running
+     PreCond:  - task is not in 'New' state.
+     PostCond: - if no timeout occurs, task is in a final
+                 state.
+     Perms:    -
      Throws:   NotImplemented
                IncorrectState
+               Timeout
                NoSuccess
      Notes:    - wait returns success (true) as soon as the
                  task enters a final state
                - if the task is already in a final state, the
                  call returns success (true) immediately.
                - if the task is in 'New' state, an
                  'IncorrectState' exception is thrown.
                - wait returns no success (false) if the task
                  is, even after timeout, not in a final state.
                - a 'Timeout' or 'NoSuccess' exception indicates
                  that the backend was not able to wait for the
```

```
                task.  Note that a 'Timeout' exception does
                not indicate that the task is not in a final
                state after the given wait period - that
                causes an unsuccessfull (false) return value.
!             - for timeout semantics, see Section 2.


    - cancel
      Purpose:  Cancel the asynchronous operation.
      Format:   cancel               (in  float  timeout);
      Inputs:   timeout:              time for freeing resources
+     InOuts:   -
      Outputs:  -
+     PreCond:  - task is in 'Running' state.
+     PostCond: - task is in 'Canceled' state.
+     Perms:    -
      Throws:   NotImplemented
                IncorrectState
+               Timeout
                NoSuccess
      Notes:    - for resource deallocation semantics, see
!                 Section 2.
                - if cancel() fails to cancel the task
!                 immediately, and tries to continue to cancel
                  the task in the background, the task state
                  remains 'Running' until the cancel operation
                  succeeded.  The state then changes to
                  'Canceled'.
                - if the task is in a final state, the call has
!                 no effect, and, in particular, does NOT change
                  the state from 'Done' to 'Canceled', or from
                  'Failed' to 'Canceled'.  This is to
                  avoid race conditions.
                - if the task is in 'New' state, an
                  'IncorrectState' exception is thrown.
                - a 'NoSuccess' exception indicates
                  that the backend was not able to initiate the
!                 cancelation for the task.
!               - for timeout semantics, see Section 2.


+   Inspection
+   ----------
+
    - get_state
      Purpose:  Get the state of the task.
```

```
      Format:   get_state            (out state state);
      Inputs:   -
+     InOuts:   -
      Outputs:  state:                state of the task.
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   NotImplemented
                Timeout
                NoSuccess
      Notes:    - a 'Timeout' or 'NoSuccess' exception indicates
                  that the backend was not able to retrieve the
                  task state.

+
+   - get_result
+     Purpose:  Get the result of the async operation
+     Format:   get_result <type>    (out type result);
+     Inputs:   -
+     InOuts:   -
+     Outputs:  result:               return value of async
+                                     method
+     PreCond:  - task is not in New, Failed or Canceled state.
+     PostCond: - task is in a final state.
+     Perms:    -
+     Throws:   NotImplemented
+               IncorrectState
+               Timeout
+               NoSuccess
+     Notes:    - get_result implies a wait() - all notes to
+                 wait apply.
+               - the method returns the type and value which
+                 would be returned by the synchronous version of
+                 the respective function call.
+
+
+   - get_object
+     Purpose:  Get the object from which this task was created
+     Format:   get_object <type>    (out type object);
+     Inputs:   -
+     InOuts:   -
+     Outputs:  object:               object this task was
+                                     created from
+     PreCond:  -
+     PostCond: -
+     Perms:    -
```

```
+     Throws:   NotImplemented
+               Timeout
+               NoSuccess
+     Notes:    - the method returns a shallow copy of the
+                 object this task was created from.
+


    - rethrow
      Purpose:  re-throw any exception a failed task caught.
      Format:   rethrow (void);
      Inputs:   -
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   NotImplemented
-               IncorrectSession
                IncorrectURL
                BadParameter
                AlreadyExists
                DoesNotExist
-               ReadOnly
-               WriteOnly
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - that method does nothing unless the task is in
                  'Failed' state, and also MUST NOT throw
                  'IncorrectState' if the task is in any other
                  state.
                - if in 'Failed' state, the method MUST raise an
                  exception which indicates the reason why that
                  task entered the 'Failed' state (i.e. it throws
                  the exception which caused it to enter the
                  'Failed' state.
+               - language bindings for languages with no
+                 support for exceptions MUST change the state
+                 of the object from which the task was created
+                 so that a susbequent call to has_error() on
+                 that object returns true.  A subsquent call to
+                 get_error() must then return the respectiv
+                 exception.
```

```
+               - rethrow can be called multiple times, always
+                 throwing the same exception.
```

Class `task_container`

Managing a large number of tasks can be tedious. The `task_container` class is *designed* to help in these situations, and to effectively handle a large number of asynchronous operations.

For example, when *an application uses* many *asynchronous* tasks, it would be inefficient to invoke the `wait()` method on each *of them individually*. The `task_container` class provides (amongst other operations) a mechanism to wait for a set of tasks.

Language bindings CAN specify the ***`task_container`*** *to be, or to inherit from, a native container type, if that allows for the same semantics as described below, and if that helps to 'naturalize' the SAGA Look & Feel for that language.*

```
    - CONSTRUCTOR
      Purpose:  create a task_container
      Format:   CONSTRUCTOR          (out task_container tc);
      Inputs:   -
+     InOuts:   -
      Outputs:  tc:                   newly created container
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   NotImplemented
                Timeout
                NoSuccess
      Notes:    - a 'Timeout' or 'NoSuccess' exception indicates
                  that the backend was not able to create a task
                  container.


    - DESTRUCTOR
      Purpose:  destroy a task_container
      Format:   DESTRUCTOR           (in task_container tc);
      Inputs:   tc:                   container to destroy
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: -
```

```
+     Perms:    -
      Throws:   -
      Notes:    - tasks in the task_container during its
                  destruction are not affected by its
                  destruction, and, in particular, are not
                  canceled.


+   Task Management
+   ---------------

    - add
      Purpose:  Add a task to a task_container.
      Format:   add                  (in  task task,
+                                     out int cookie);
      Inputs:   task:                task to add to the
                                     task_container
+     InOuts:   -
+     Outputs:  cookie:              cookie identifying the
+                                     added task
+     PreCond:  -
+     PostCond: - the task is managed by the task container.
+     Perms:    -
      Throws:   NotImplemented
                Timeout
                NoSuccess
!     Notes:    - a task can be added only once.  Any attempt
+                 to add a task to the container which already
+                 is in the container is silently ignored, and
+                 the same cookie as for the original task is
+                 returned again.
                - a 'Timeout' or 'NoSuccess' exception indicates
                  that the backend was not able to add the task
                  to the container.


    - remove
      Purpose:  Remove a task from a task_container.
      Format:   remove               (in  int  cookie,
                                     out task task);
      Inputs:   task:                cookie identifying the
                                     task to be removed
+     InOuts:   -
      Outputs:  task:                the removed task
+     PreCond:  - the task is managed by the task container.
+     PostCond: - the task is not managed by the task container.
```

```
+     Perms:    -
      Throws:   NotImplemented
                DoesNotExist
                Timeout
                NoSuccess
!     Notes:    - if a task was added more than once, it can be
!                 removed only once - see notes to add().
!               - if the task identified by the cookie is not in
                  the task_container, a 'DoesNotExist' exception
                  is thrown.
                - a 'Timeout' or 'NoSuccess' exception indicates
                  that the backend was not able to remove the
                  task from the container.


+   State Management
+   ----------------

    - run
      Purpose:  Start all asynchronous operations in the
                container.
      Format:   run (void);
      Inputs:   -
+     InOuts:   -
      Outputs:  -
+     PreCond:  - all tasks in the comtainer are in 'New' state.
+     PostCond: - all tasks in the comtainer are in 'Running'
+                 state.
+     Perms:    - see permissions on task::run()
      Throws:   NotImplemented
                IncorrectState
+               DoesNotExist
                Timeout
                NoSuccess
      Notes:    - run() MUST cause an 'IncorrectState' exception
                  if any of the tasks in the container causes
                  that exception on run().
                - a 'Timeout' or 'NoSuccess' exception indicates
                  that the backend was not able to run one or
                  more tasks in the container.
+               - if the task_container is empty, an
+                 'DoesNotExist' exception is thrown.
                - As the order of execution of the tasks is
                  undefined, no assumption on the individual
                  task states can be made after any
                  exception gets thrown.
```

```
    - wait
      Purpose:  Wait for one or more of the tasks to finish.
!     Format:   wait                 (in  wait_mode mode = All,
!                                     in  float  timeout = -1.0,
                                      out task   done);
!     Inputs:   mode:                 wait for All or Any task
!               timeout:              seconds to wait
+     InOuts:   -
      Outputs:  done:                 finished task
+     PreCond:  -
+     PostCond: - if no timeout occurs, All/Any tasks in the
+               container are in a final state.
+     Perms:    -
      Throws:   NotImplemented
+               IncorrectState
+               DoesNotExist
+               Timeout
                NoSuccess
      Notes:    - if mode is 'All', the wait call returns only
                  if all tasks in the container are finished,
!                 or on timeout, whichever occurs first.
                  The output task is then any of the finished
                  tasks.
                - if mode is 'Any', the wait call returns on the
                  first task which would return on task::wait in
                  that timeout period, and returns that task.
                - the default wait mode is 'All' (0).
                - the returned task is removed from the
                  container, which allows constructs like
!                   while ( tc.size () )
                    {
!                      saga::task t = tc.wait (saga::task::Any) )
                       ...
                    }
!               - wait() MAY cause an 'IncorrectState' exception
                  if any of the tasks in the container causes
                  that exception on wait().
+               - if the task_container is empty, an
+                 'DoesNotExist' exception is thrown.
                - a 'Timeout' or 'NoSuccess' exception indicates
                  that the backend was not able to wait for one
                  or more tasks in the container.
                - As the order of execution of the tasks is
                  undefined, no assumption on the individual
```

```
                  task states can be made after any
                  exception gets thrown.
!               - for timeout semantics, see Section 2.


    - cancel
      Purpose:  Cancel all the asynchronous operations in the
                container.
      Format:   cancel               (in  float  timeout);
      Inputs:   timeout:              time for freeing resources
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - if no timeout occurs, all tasks in the
+                 container are in 'Canceled' state.
+     Perms:    -
!     Throws:   NotImplemented
+               IncorrectState
+               DoesNotExist
+               Timeout
+               NoSuccess
      Notes:    - see semantics of task cancel.
!               - cancel() MUST cause an 'IncorrectState'
!                 exception if any of the tasks in the container
!                 causes that exception on cancel().
                - a 'Timeout' or 'NoSuccess' exception indicates
                  that the backend was not able to run one or
                  more tasks in the container.
+               - if the task_container is empty, an
+                 'DoesNotExist' exception is thrown.
                - As the order of execution of the tasks is
                  undefined, no assumption on the individual
                  task states can be made after any
                  exception gets thrown.


+   Inspection
+   ----------

+   - size
+     Purpose:  return the number of tasks in the task
+               task_container.
+     Format:   size                 (out int  n);
+     Inputs:   -
+     InOuts:   -
+     Outputs:  n:                    number of tasks in
```

```
+                                     task_container
+     PreCond:  -
+     PostCond: -
+     Perms:    -
+     Throws:   NotImplemented
+               Timeout
+               NoSuccess
+     Notes:    - a 'Timeout' or 'NoSuccess' exception indicates
+                 that the backend was not able to list the
+                 tasks in the container.


    - list_tasks
!     Purpose:  List the tasks in the task_container.
      Format:   list_tasks           (out array<int>   cookies);
+     Inputs:   -
+     InOuts:   -
      Outputs:  cookies:              array of cookies for all
                                      tasks in task_container
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   NotImplemented
                Timeout
                NoSuccess
      Notes:    - a 'Timeout' or 'NoSuccess' exception indicates
                  that the backend was not able to list the
                  tasks in the container.

+
+   - get_task
+     Purpose:  Get a single task from the task_container.
+     Format:   get_task             (in  int      cookie,
+                                     out task     t);
+     Inputs:   cookie:               the cookie identifying the
+                                     task to return
+     InOuts:   -
+     Outputs:  t:                    the task identified by
+                                     cookie
+     PreCond:  -
+     PostCond: -
+     Perms:    -
+     Throws:   NotImplemented
+               DoesNotExist
+               Timeout
+               NoSuccess
```

```
+     Notes:    - the returned task is NOT removed from the
+                 task_container.
+               - if cookie specifies a task which is not in the
+                 container, a 'DoesNotExist' exception is
+                 thrown.
+               - a 'Timeout' or 'NoSuccess' exception indicates
+                 that the backend was not able to list the
+                 tasks in the container.
+

    - get_tasks
!     Purpose:  Get the tasks in the task_container.
      Format:   get_tasks            (out array<task>   tasks);
+     Inputs:   -
+     InOuts:   -
      Outputs:  tasks:                array of tasks in
                                      task_container
+     PreCond:  -
+     PostCond: -
+     Perms:    -
!     Throws:   NotImplemented
+               Timeout
+               NoSuccess
      Notes:    - the returned tasks are NOT removed from the
!                 task_container.
+               - if the task_container is empty, an
+                 empty list is returned.
                - a 'Timeout' or 'NoSuccess' exception indicates
                  that the backend was not able to list the
                  tasks in the container.


    - get_states
!     Purpose:  Get the states of all tasks in the
                task_container.
      Format:   get_states           (out array<state>   states);
+     Inputs:   -
+     InOuts:   -
      Outputs:  states:               array of states for
                                      tasks in task_container
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   NotImplemented
                Timeout
                NoSuccess
```

```
      Notes:    - the returned list is not ordered
+               - if the task_container is empty, an
+                 empty list is returned.
                - a 'Timeout' or 'NoSuccess' exception indicates
                  that the backend was not able to obtain the
                  states of the tasks in the container.
```

3.10.4 Examples

Code Example

```
1  // c++ example
2  saga::directory dir;
3  saga::job       job;
4
5  ...
6
7  /* create tasks */
8  saga::task t1 = dir.ls         <saga::task> (result);
9  saga::task t2 = dir.copy       <saga::task> (source,target);
10 saga::task t3 = dir.move       <saga::task> (source,target);
11 saga::task t4 = job.checkpoint <saga::task> ();
12 saga::task t5 = job.signal     <saga::task> (SIG_USR);
13
14 // start tasks
15 t1.run ();
16 t2.run ();
17 t3.run ();
18 t4.run ();
19 t5.run ();
20
21 // put all tasks into container
22 saga::task_container tc;
23
24 tc.add (t1);
25 tc.add (t2);
26 tc.add (t3);
27 tc.add (t4);
28 tc.add (t5);
29
30 // take one out again
31 tc.remove (t5);
32
33 // wait for all other tasks in container to finish
34 tc.wait ();
35
36 // wait for the last task
37 t5.wait ();
```

```
38
39  +-------------------------------------------------------------+
40
41  // example for error handling in C++
42  {
43    task.run  ();
44    task.wait ();
45
46    if ( task.get_state () == saga::task::Failed )
47    {
48      try {
49        task.rethrow ();
50      }
51      catch ( const saga::exception & e )
52      {
53        std::cout << "task failed: "
54                  << e.get_message ()
55                  << std::endl;
56      }
57    }
58  }
```

4 SAGA API Specification – API Packages

The Functional SAGA API packages define the functional SAGA API scope, as motivated in the **Introduction** and in [18].

General Properties of Functional API Classes and Instances

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

Nearly all of the SAGA use cases (except for the GridRPC use cases) had either explicit or implicit requirements for submitting jobs to grid resources, and most needed to also to monitor and control these submitted jobs.

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.

This API is also intended to incorporate the work of the DRMAA-WG [9]. Much of this specification was taken directly from DRMAA specification [24], with many of the differences arising from an attempt to make the job API consistent with the overall SAGA API **Look-&-Feel**[3].

The API covers four classes: `saga::job_description`, `saga::job_service`, `saga::job` and `saga::job_self`. The job description class is nothing more than a container for a well defined set of attributes which, using JSDL [15] 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 *insection* of jobs.

The job class itself is central to the API, and represents an application instance running under the management of a resource manager. The `job_self` class `IS-A` job, but additionally implements the steering interface. The purpose of this class 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 class inherits the `saga::task` class 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 `saga::job` class 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. In this specification, the standard I/O streams are specified to have `opaque` types. The SAGA language bindings MUST specify a native type for I/O streams. That type SHOULD be the one used as the file descriptor to the POSIX `read()` call in that language.

[3] We expect that SAGA-API implementations may be implemented using DRMAA, or may produce JSDL documents to be passed to underlying scheduling systems.

4.1.1 Job State Model

The SAGA job state diagram is shown in Figure 4. It is an extension of the `saga::task` state diagram (Figure 3), and extends the state diagram *with an 'Unknown' state (which is needed for job instances which are not yet initialized, and are to be used for asynchronous initialization), and* with a `'Suspended'` state, which the job can enter/leave using the `suspend()/resume()` calls.

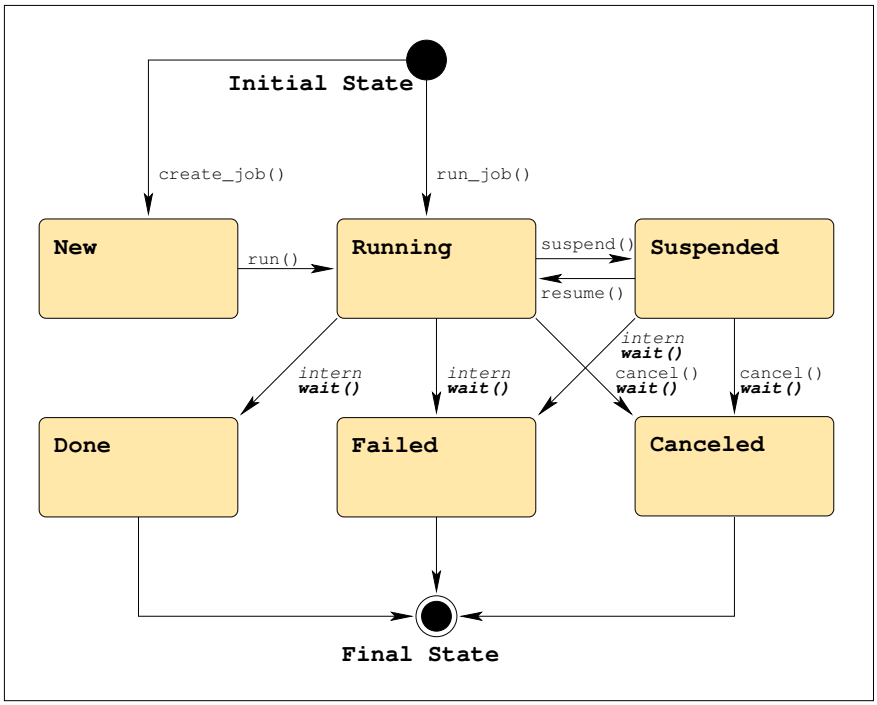

Figure 4: The SAGA job state model extends the SAGA task state model with *'Unknown' and* a 'Suspended' state, and additional transitions (See Figure 1 for a legend).

SAGA implementations need to map the native backend state model onto the SAGA state model. The SAGA state model should be simple enough to allow a straight forward mapping in most cases. For some applications, access to the native backend state model is useful. For that reason, an additional metric named `'StateDetail'` allows to query the native job state. That schema follows the current state model of the OGSA-BES specification [12], which also has a simplified top level state model, and allows for additional, backend specific state details.

State details in SAGA SHOULD be formatted as follows:

```
'<model>:<state>'
```

with valid models being "BES", "DRMAA", or other implementation specific models. For example, a state detail for the BES state '**StagingIn**' would be rendered as `'BES:StagingIn'`)*, and would be a substate of* ***Running***. If no state details are available, the metric is still available, but it has always an empty string value.

4.1.2 Job Description Attributes

SAGA implementations MUST support the `Executable` attribute, as that is the only required attribute for a `job_description`. An implementation MUST document which other attributes are supported, and which **are not**. In general, a `job_description` containing an unsupported attribute does *not* cause an error on job creation or submission, unless noted otherwise in the attribute description.

Attributes marked as `'not supported by JSDL'` might disappear in future versions of the SAGA API – all other attributes are likely to be kept, at least for backward compatibility. The attribute description additionally mentions if the attributes are supported by DRMAA (see [24]) – that is for information purposes only, and supposed to support implementations on top of DRMAA.

Several Metrics on the `saga::job` class (the class implements the `saga::monitorable` interface) reflect attributes from the job description. This redundancy is intentional, and aims at providing information about (a) attributes which may change at runtime, and (b) attributes for jobs for which no job description is available (e.g. `saga::job` instances obtained by calling `get.job()`.

Although JSDL [3] *and JSDL SPMD extension [8]* based attribute names are used for job description, the API *supports* no explicit representation of JSDL (i.e. JSDL compliant XML). XML is deemed to be too low level to be included into the SAGA API. *Also, the JSDL parameter sweep extension [7] is not used in SAGA at the moment, as bulk job submission, and related the creation of multiple related job descriptions, is performed on application level in SAGA, as described in section 2.9.*

4.1.3 File Transfer Specifications

The syntax of a file transfer directive for the job description is modeled on the LSF syntax *(LSF stands for Load Sharing Facility, a commercial job scheduler by Platform Computing)*, and has the general syntax:

```
local_file operator remote_file
```

Both the `local_file` and the `remote_file` can be URLs. If they are not URLs,

but full or relative pathnames, then the `local_file` is relative to the host where the submission is executed, and the `remote_file` is evaluated on the execution host of the job.

The operator is one of the following four:

`'>'`	copies the local file to the remote file before the job starts. Overwrites the remote file if it exists.
`'>>'`	copies the local file to the remote file before the job starts. Appends to the remote file if it exists.
`'<'`	copies the remote file to the local file after the job finishes. Overwrites the local file if it exists.
`'<<'`	copies the remote file to the local file after the job finishes. Appends to the local file if it exists.

4.1.4 Command Line Specification

The `run_job()` method of the `saga::job_service` class accepts a string parameter which constitutes a command line to be executed on a remote resource. The parsing of that command lines follows the following rules:

- **Elements** are delimited by white space, which is either a space or a tab.
- A string surrounded by double quotation marks is interpreted as a single element, regardless of white space contained within. A quoted string can be embedded in an element.
- A double quotation mark preceded by a backslash, `\"`, is interpreted as a literal double quotation mark (`"`).
- Backslashes are interpreted literally, unless they immediately precede a double quotation mark.
- The first **element** is used as executable name; all other elements are treated as job arguments.

4.1.5 Job Identifiers

The `JobID` is treated as an opaque string in the SAGA API. However, for the sake of interoperability of different SAGA implementations, and for potential extended use of the `JobID` information, the `JobID` SHOULD be implemented as:

```
'[backend url]-[native id]'
```

For example, a job submitted to the host `remote.host.net` via `ssh` (whose daemon runs on port 22), and having the *POSIX* **PID** `1234`, should get the job id:

```
'[ssh://remote.host.net:22/]-[1234]'
```

The implementation MAY free the resources used for the job, and hence MAY invalidate a `JobID`, after a successful wait on the job, or after the application **received** the job status information, and job status details if available, at least once.

*A JobID may be unknown until the job enters the **`Running`** state, as the backend will often not assign IDs to jobs which are not yet running. In such cases, the value of the **`JobID`** attribute SHOULD be empty. The job MUST, however, retain its JobID after it enters in a final state.*

4.1.6 Specification

```
  package saga.job
  {
    enum state
    {
-     Unknown    = -1,  // same as in saga::task::state
      New        =  1,  // same as in saga::task::state
      Running    =  2,  // same as in saga::task::state
      Done       =  3,  // same as in saga::task::state
      Canceled   =  4,  // same as in saga::task::state
      Failed     =  5,  // same as in saga::task::state
      Suspended  =  6
    }


    class job_description : implements   saga::object
                            implements   saga::attributes
                          // from object: saga::error_handler
    {
      CONSTRUCTOR          (out job_description obj);
      DESTRUCTOR           (in  job_description obj);

      // Attributes:
      //   name:  Executable
      //   desc:  command to execute.
      //   type:  String
```

```
      //   mode:  ReadWrite
      //   value: ''
      //   notes: - this is the only required attribute.
      //          - can be a full pathname, or a pathname
      //            relative to the 'WorkingDirectory' as
      //            evaluated on the execution host.
      //          - semantics as defined in JSDL
      //          - available in JSDL, DRMAA
      //
      //   name:  Arguments
      //   desc:  positional parameters for the command.
      //   mode:  ReadWrite, optional
      //   type:  Vector String
      //   value: -
      //   notes: - semantics as specified by JSDL
      //          - available in JSDL, DRMAA
      //
+     //   name:  SPMDVariation
+     //   desc:  SPMD job type and startup mechanism
+     //   mode:  ReadWrite, optional
+     //   type:  String
+     //   value: -
+     //   notes: - as defined in the SPMD extension of JSDL
+     //   notes: - semantics as defined in JSDL
+     //          - available in JSDL, SPMD extension
+     //          - the SPMD JSDL extension defines the value
+     //            to be an URI.  For simplicity, SAGA allows
+     //            the following strings, which map into the
+     //            respective URIs: MPI, GridMPI, IntelMPI,
+     //            LAM-MPI, MPICH1, MPICH2, MPICH-GM, MPICH-MX,
+     //            MVAPICH, MVAPICH2, OpenMP, POE, PVM, None
+     //          - the value '' (no value, default) indicates
+     //            that the application is not a SPMD
+     //            application.
+     //          - as JSDL, SAGA allows other arbitrary values.
+     //            The implementation must clearly document
+     //            which values are supported.
+     //
      //   name:  TotalCPUCount
      //   desc:  total number of cpus requested for this job
      //   mode:  ReadWrite, optional
      //   type:  Int
      //   value: '1'
      //   notes: - semantics as defined in JSDL
      //          - available in JSDL, DRMAA
      //
```

```
+     //   name:  NumberOfProcesses
+     //   desc:  total number of processes to be started
+     //   mode:  ReadWrite, optional
+     //   type:  Int
+     //   value: '1'
+     //   notes: - semantics as defined in JSDL
+     //          - available in JSDL, SPMD extension
+     //
+     //   name:  ProcessesPerHost
+     //   desc:  number of processes to be started per host
+     //   mode:  ReadWrite, optional
+     //   type:  Int
+     //   value: '1'
+     //   notes: - semantics as defined in JSDL
+     //          - available in JSDL, SPMD extension
+     //
+     //   name:  ThreadsPerProcess
+     //   desc:  number of threads to start per process
+     //   mode:  ReadWrite, optional
+     //   type:  Int
+     //   value: '1'
+     //   notes: - semantics as defined in JSDL
+     //          - available in JSDL, SPMD extension
+     //
      //   name:  Environment
      //   desc:  set of environment variables for the job
      //   mode:  ReadWrite, optional
      //   type:  Vector String
      //   value: -
      //   notes: - exported into the job environment
      //          - format: 'key=value'
      //          - semantics as specified by JSDL
!     //          - available in JSDL, DRMAA
      //
      //   name:  WorkingDirectory
      //   desc:  working directory for the job
      //   mode:  ReadWrite, optional
      //   type:  String
      //   value: '.'
      //   notes: - semantics as specified by JSDL
      //          - available in JSDL, DRMAA
      //
      //   name:  Interactive
      //   desc:  run the job in interactive mode
      //   mode:  ReadWrite, optional
      //   type:  Bool
```

```
      //   value: 'False'
      //   notes: - this implies that stdio streams will stay
      //            connected to the submitter after job
      //            submission, and during job execution.
      //          - if an implementation cannot handle
!     //            interactive jobs, and this attribute is
      //            present, and 'True', the job creation MUST
!     //            throw an 'IncorrectParameter' error with a
      //            descriptive error message.
      //          - not supported by JSDL, DRMAA
      //
      //   name:  Input
      //   desc:  pathname of the standard input file
      //   mode:  ReadWrite, optional
      //   type:  String
      //   value: -
      //   notes: - semantics as specified by JSDL
      //          - available in JSDL, DRMAA
+     //          - will not be used if 'Interactive' is 'True'
      //
      //   name:  Output
      //   desc:  pathname of the standard output file
      //   mode:  ReadWrite, optional
      //   type:  String
      //   value: -
      //   notes: - semantics as specified by JSDL
      //          - available in JSDL, DRMAA
+     //          - will not be used if 'Interactive' is 'True'
      //
      //   name:  Error
      //   desc:  pathname of the standard error file
      //   mode:  ReadWrite, optional
      //   type:  String
      //   value: -
      //   notes: - semantics as specified by JSDL
      //          - available in JSDL, DRMAA
+     //          - will not be used if 'Interactive' is 'True'
      //
-     //   name:  JobName
-     //   desc:  job name to be attached to the job submission
-     //   mode:  ReadWrite, optional
-     //   type:  String
-     //   value: 'False'
-     //   notes: - available in DRMAA
-     //          - not supported by JSDL
-     //
```

```
      //   name:  FileTransfer
      //   desc:  a list of file transfer directives
      //   mode:  ReadWrite, optional
      //   type:  Vector String
      //   value: -
      //   notes: - translates into jsdl:DataStaging
      //          - used to specify pre- and post-staging
      //          - semantics as specified in JSDL
+     //          - staging is part of the 'Running' state
      //          - syntax similar to LSF (see earlier notes)
      //          - available in JSDL, DRMAA
      //
      //   name:  Cleanup
!     //   desc:  defines if output files get removed after the
      //          job finishes
      //   mode:  ReadWrite, optional
      //   type:  String
      //   value: 'Default'
      //   notes: - can have the Values 'True', 'False', and
      //            'Default'
      //          - On 'False', output files MUST be kept
!     //            after job the finishes
      //          - On 'True', output files MUST be deleted
!     //            after job the finishes
      //          - On 'Default', the behaviour is defined by
      //            the implementation or the backend.
      //          - translates into 'DeleteOnTermination' elements
      //            in JSDL
      //
      //   name:  JobStartTime
!     //   desc:  time at which a job should be scheduled
      //   mode:  ReadWrite, optional
      //   type:  Int
      //   value: -
      //   notes: - Could be viewed as a desired job start
      //            time, but that is up to the resource
      //            manager.
      //          - format: number of seconds since epoch
      //          - available in DRMAA
      //          - not supported by JSDL
      //
-     //   name:  Deadline
-     //   desc:  hard deadline after which the resource
-     //          manager SHOULD cancel the job.
-     //   mode:  ReadWrite, optional
-     //   type:  Int
```

```
-     //   value: -
-     //   notes: - Could be viewed as a desired job start
-     //            time, but that is up to the resource
-     //            manager.
-     //   notes: - format: number of seconds since epoch
-     //          - available in DRMAA
-     //          - not supported by JSDL
-     //
-     //   name:  WallTimeLimit
-     //   desc:  hard limit on the amount of wall clock time
-     //          in seconds that a job may consume
-     //   mode:  ReadWrite, optional
-     //   type:  Int
-     //   value: -
-     //   notes: - semantics as defined in JSDL
-     //          - available in JSDL, DRMAA
-     //
!     //   name:  TotalCPUTime
!     //   desc:  estimate total number of CPU seconds which
!     //          the job will require.
      //   mode:  ReadWrite, optional
      //   type:  Int
      //   value: -
      //   notes: - intended to provide hints to the scheduler.
-     //          - if the limit is reached, the action taken is
-     //            specific to the resource manager and its
      //            scheduling policies.
      //          - available in JSDL, DRMAA
+     //          - semantics as defined in JSDL
      //
-     //   name:  CPUTimeLimit
-     //   desc:  estimated job runtime in CPU seconds.
-     //   mode:  ReadWrite, optional
-     //   type:  Int
-     //   value: -
-     //   notes: - semantics as defined in JSDL
-     //          - available in JSDL, DRMAA
-     //
      //   name:  TotalPhysicalMemory
      //   desc:  Estimated amount of memory the job requires
      //   mode:  ReadWrite, optional
      //   type:  Float
      //   value: -
      //   notes: - unit is in MegaByte
      //          - memory usage of the job is aggregated
      //            across all processes of the job
```

```
      //          - semantics as defined by JSDL
!     //          - available in JSDL
      //
      //   name:  CPUArchitecture
      //   desc:  compatible processor for job submission
      //   mode:  ReadWrite, optional
      //   type:  Vector String
      //   value: -
      //   notes: - allowed values as specified in JSDL
      //          - semantics as defined by JSDL
!     //          - available in JSDL
      //
      //   name:  OperatingSystemType
      //   desc:  compatible operating system for job submission
      //   mode:  ReadWrite, optional
      //   type:  Vector String
      //   value: -
      //   notes: - allowed values as specified in JSDL
      //          - semantics as defined by JSDL
!     //          - available in JSDL
      //
      //   name:  CandidateHosts
!     //   desc:  list of host names which are to be considered
      //          by the resource manager as candidate targets
      //   mode:  ReadWrite, optional
      //   type:  Vector String
      //   value: -
      //   notes: - semantics as defined by JSDL
!     //          - available in JSDL
      //
      //   name:  Queue
      //   desc:  name of a queue to place the job into
      //   mode:  ReadWrite, optional
      //   type:  String
      //   value: -
      //   notes: - While SAGA itself does not define the
!     //            semantics of a "queue", many backend systems
      //            can make use of this attribute.
      //          - not supported by JSDL
      //
      //   name:  JobContact
      //   desc:  set of endpoints describing where to report
      //          job state transitions.
      //   mode:  ReadWrite, optional
      //   type:  Vector String
      //   value: -
```

```
      //   notes: - format: URI (e.g. fax:+123456789,
      //             sms:+123456789, mailto:joe@doe.net).
      //          - available in DRMAA
      //          - not supported by JSDL
    }


    class job_service : implements   saga::object
                        implements   saga::async
                      // from object  saga::error_handler
    {
!     CONSTRUCTOR          (in  session         s,
!                           in  url             rm = "",
!                           out job_service     obj);
!     DESTRUCTOR           (in  job_service     obj);

      create_job           (in  job_description jd,
                            out job             job);
!     run_job              (in  string          commandline,
!                           in  string          host = "",
                            out job             job,
                            out opaque          stdin,
                            out opaque          stdout,
                            out opaque          stderr);
      list                 (out array<string>   job_ids);
      get_job              (in  string          job_id,
                            out job             job);
      get_self             (out job_self        job);
    }


    class job : extends      saga::task
                implements   saga::async
                implements   saga::attributes
+               implements   saga::permissions
              // from task    saga::object
              // from task    saga::monitorable
              // from object  saga::error_handler
    {
-     CONSTRUCTOR          (void                   );
+     // no CONSTRUCTOR
      DESTRUCTOR           (in  job             obj);

      // job inspection
      get_job_description  (out job_description jd);
      get_stdin            (out opaque          stdin);
```

```
      get_stdout           (out opaque          stdout);
      get_stderr           (out opaque          stderr);

      // job management
      suspend              (void);
      resume               (void);
      checkpoint           (void);
      migrate              (in job_description   jd);
      signal               (in int               signum);

      // Attributes:
      //   name:  JobID
      //   desc:  SAGA representation of the job identifier
      //   mode:  ReadOnly
      //   type:  String
      //   value: -
      //   notes: - format: as described earlier
      //
      //   name:  ExecutionHosts
      //   desc:  list of host names or IP addresses allocated
      //          to run this job
      //   mode:  ReadOnly, optional
      //   type:  Vector String
      //   value: -
      //   notes: -
      //
      //   name:  Created
      //   desc:  time stamp of the job creation in the
      //          resource manager
      //   mode:  ReadOnly, optional
      //   type:  Time
      //   value: -
!     //   notes: - can be interpreted as submission time
      //
      //   name:  Started
      //   desc:  time stamp indicating when the job started
      //          running
      //   mode:  ReadOnly, optional
      //   type:  Time
      //   value: -
      //
      //   name:  Finished
      //   desc:  time stamp indicating when the job completed
      //   mode:  ReadOnly, optional
      //   type:  Time
      //   value: -
```

```
      //
      //   name:  WorkingDirectory
      //   desc:  working directory on the execution host
      //   mode:  ReadOnly, optional
      //   type:  String
      //   value: -
      //   notes: - can be used to determine the location of
      //            files staged using relative file paths
      //
      //   name:  ExitCode
      //   desc:  process exit code as collected by the wait(2)
      //          series of system calls.
      //   mode:  ReadOnly, optional
      //   type:  Int
      //   value: -
      //   notes: - exit code is collected from the process
      //            which was started from the 'Executable'
      //            attribute of the job_description object.
      //          - only available in final states, if at all
      //
      //   name:  Termsig
      //   desc:  signal number which caused the job to exit
      //   mode:  ReadOnly, optional
      //   type:  Int
      //   value: -
      //   notes: - only available in final states, if at all


      // Metrics:
!     //   name:  job.state
      //   desc:  fires on state changes of the job, and has
      //          the literal value of the job state enum.
      //   mode:  ReadOnly
      //   unit:  1
      //   type:  Enum
!     //   value: New
      //   notes: - the state metric is inherited from
      //            saga::task, but has a different set
      //            of possible values
      //          - see description of job states above
      //
!     //   name:  job.state_detail
      //   desc:  fires as a job changes its state detail
      //   mode:  ReadOnly, optional
      //   unit:  1
      //   type:  String
```

```
      //   value: -
      //
!     //   name:  job.signal
      //   desc:  fires as a job receives a signal, and has a
      //          value indicating the signal number
      //   mode:  ReadOnly, optional
      //   unit:  1
      //   type:  Int
      //   value: -
      //   notes: - no guarantees are made that any or all
      //            signals can be notified by this metric
      //
-     //   name:  CPUTimeLimit
+     //   name:  job.cpu_time
!     //   desc:  number of CPU seconds consumed by the job
      //   mode:  ReadOnly, optional
      //   unit:  seconds
      //   type:  Int
      //   value: -
      //   notes: - aggregated across all processes/threads
      //
!     //   name:  job.memory_use
      //   desc:  current aggregate memory usage
      //   mode:  ReadOnly, optional
      //   unit:  megabyte
      //   type:  Float
!     //   value: 0.0
!     //   notes: - metric becomes 'Final' after job
!     //            completion, and then shows the memory
      //            high water mark
      //
!     //   name:  job.vmemory_use
      //   desc:  current aggregate virtual memory usage
      //   mode:  ReadOnly, optional
      //   unit:  megabyte
      //   type:  Float
!     //   value: 0.0
!     //   notes: - metric becomes 'Final' after job
!     //            completion, and then shows the virtual
      //            memory high water mark
      //
!     //   name:  job.performance
      //   desc:  current performance
      //   mode:  ReadOnly, optional
      //   unit:  FLOPS
      //   type:  Float
```

```
!     //   value: 0.0
!     //   notes: - metric becomes 'Final' after job
!     //            completion, and then shows the performance
      //            high water mark
    }


    class job_self : extends     saga::job
                     implements  saga::steerable
                  // from job    saga::async
                  // from job    saga::attributes
                  // from job    saga::task
                  // from job    saga::object
                  // from job    saga::monitorable
                  // from job    saga::permissions
                  // from job    saga::error_handler
    {
      // no CONSTRUCTOR
      DESTRUCTOR           (in  job_self        obj);
    }
  }
```

4.1.7 Specification Details

Enum `state`

The ***`state`*** *is equivalent to the inherited* ***`saga::task::state`****, but adds the* ***`Suspended`*** *state:*

`Suspended`

> *This state identifies a job instance which has been suspended. This state corresponds to the BES state* ***`'Suspend'`****.*

Class `job_description`

This object encapsulates all the attributes which define a job to be run. It has no methods of its own, but implements the `saga::attributes` interface in order to provide access to the job properties, which are expressed as JSDL keywords.

The only required attribute in order to perform a valid job submission is the `Executable`. Given the `Executable`, a job can be **instantiated** in many existing backend systems without any further specification.

There should be **significant** overlap between the attributes defined within SAGA and within the JSDL specification. This list, however, will not be complete in cases where the JSDL was deemed more complicated than was required for a simple API (e.g. the notion of JSDL **profiles**), or where an attribute was needed to interact with a scheduler, which was not within the stated scope of the JSDL working group (e.g. `Queue`, which is considered a *site attribute*, and thus not relevant to the pure description of a job).

```
    - CONSTRUCTOR
      Purpose:  create the object
      Format:   CONSTRUCTOR       (out job_description obj)
      Inputs:   -
+     InOuts:   -
      Outputs:  obj:               the newly created object
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   NotImplemented
+               NoSuccess
      Notes:    - a job_description is not associated with a
                  session, but can be used for job services
                  from different sessions.


    - DESTRUCTOR
      Purpose:  destroy the object
      Format:   DESTRUCTOR        (in  job_description obj)
      Inputs:   obj:               the object to destroy
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   -
      Notes:    -
```

Class `job_service`

The `job_service` represents a resource management backend, and as such allows to create and submit jobs, and to discover jobs. The job management methods are on the job object itself – **this** probably implies that implementations need to internally track what resource manager (or `job_service` *instance*)

created the job.

```
    - CONSTRUCTOR
      Purpose:  create the object
!     Format:   CONSTRUCTOR       (in  session       s,
!                                  in  url           rm = "",
                                   out job_service   obj)
!     Inputs:   s:                session to associate with
                                   the object
!               rm:               contact url for resource
                                   manager
+     InOuts:   -
      Outputs:  obj:              the newly created object
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   NotImplemented
                IncorrectURL
-               DoesNotExist
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - 'rm' defaults to an empty string - in that
                  case, the implementation must perform a
                  resource discovery, or fall back to a fixed
                  value, or find a valid  rm contact in any
                  other way.  If that is not possible, a
                  'BadParameter' exception MUST be thrown, and
                  MUST indicate that a rm contact string is
                  needed.  The expected behaviour MUST be
                  documented (i.e. if a default is available).
-               - if the URL's schema cannot be used by the
-                 implementation, an 'IncorrectURL' exception
-                 is thrown.
                - if the rm identified by the rm URL cannot be
                  contacted (i.e. does not exist), a
                  'BadParameter' exception is thrown.


    - DESTRUCTOR
      Purpose:  destroy the object
      Format:   DESTRUCTOR        (in  job_service   obj)
```

```
      Inputs:   obj:               the object to destroy
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - jobs created by that job_service instance
+                 are not affected by the destruction, and are
+                 in particular not canceled.
+     Perms:    -
      Throws:   -
      Notes:    -


    - create_job
      Purpose:  create a job instance
!     Format:   create_job        (in  job_description jd,
                                    out job             job);
!     Inputs:   jd:                description of job to be
                                    submitted
+     InOuts:   -
      Outputs:  job:               a job object representing
                                    the submitted job instance
+     PreCond:  - jd has an 'Executable' attribute.
+     PostCond: - job is in 'New' state
+               - jd is deep copied (no state is shared
+                 after method invocation)
+               - 'Owner' of the job is the id of the context
+                 used for creating the job.
+     Perms:    -
+     Throws:   NotImplemented
+               BadParameter
+               PermissionDenied
+               AuthorizationFailed
+               AuthenticationFailed
+               Timeout
+               NoSuccess
      Notes:    - calling run() on the job will submit it to
                  the resource, and advance its state.
+               - if the job description does not have a valid
+                 'Executable' attribute, a 'BadParameter'
+                 exception is thrown.
                - if the job description contains values which
                  are outside of the allowed range, or cannot be
                  parsed, or are otherwise invalid and not
                  usable for creating a job instance, a
                  'BadParameter' exception is thrown, which MUST
!                 indicate which attribute(s) caused this
```

```
                  exception, and why.


    - run_job
      Purpose:  Run a command synchronously.
!     Format:   run_job           (in  string  commandline,
!                                  in  string  host = "",
                                   out job     job,
                                   out opaque  stdin,
                                   out opaque  stdout,
                                   out opaque  stderr);
!     Inputs:   commandline:       the command and arguments
!                                  to be run
!               host:              hostname to be used by rm for
!                                  submission
+     InOuts:   -
      Outputs:  stdin:             IO handle for the running
                                   job's standard input stream
                stdout:            IO handle for the running
                                   job's standard output
                stderr:            IO handle for the running
                                   job's standard error
                job:               a job object representing
                                   the submitted job instance
+     PreCond:  -
+     PostCond: - job is in 'Running', 'Done' or 'Failed' state.
+               - 'Owner' of the job is the id of the context
+                 used for creating the job.
+     Perms:    -
      Throws:   NotImplemented
                BadParameter
-               IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - This is a convenience routine built on the
                  create_job method, and is intended to simplify
                  the steps of creating a job_description,
                  creating and running the job, and then
                  querying the standard I/O streams.
                - the I/O handles have to be passed to the call
                  as references, in most languages, as calls
                  often allow only one return value (perl or
!                 python being notable exceptions).  If these
```

```
!                 parameters are omitted, the job is to be
                  started non-interactively, and the output I/O
                  streams may be discarded.
                - the job is guaranteed to run on the given
                  host, or not at all.
                - the method is exactly equivalent to the
                  sequence of (1) creation of a job_description
-                 with 'Executable'/Environment set to the
-                 values from
+                 with 'Executable' set to the values from the
                  commandline, 'Interactive' set if I/O is
                  requested, 'CandidateHost' set to host; (2)
                  create_job() with that description; (3)
                  calling run() on that job.  This method can
                  throw any of the exceptions which can occur in
                  this sequence, with the semantics defined in
                  the detailed description of the methods used
                  in this sequence.  No other exception are to
                  be expected.
                - if 'host' is an empty string (the default),
!                 the implementation MUST choose an arbitrary
                  host for execution.
+               - stdin, stdout and stderr are guaranteed to
+                 contain/provide the complete standard I/O
+                 streams, beginning at the start of the remote
+                 process.


    - list
      Purpose:  Get a list of jobs which are currently known by
                the resource manager.
      Format:   list              (out array<string>   job_ids);
      Inputs:   -
+     InOuts:   -
      Outputs:  job_ids:           an array of job identifiers
+     PreCond:  -
+     PostCond: -
+     Perms:    Query on jobs identified by the returned ids
      Throws:   NotImplemented
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
+     Notes:    - which jobs are viewable by the calling user
+               context, and how long a resource manager keeps
```

```
+                job information, are both implementation
+                dependent.
               - a returned job_id may translate into a job
                 (via get_job()) which is not controllable by
                 the requesting application (e.g. it could
!                cause an 'AuthorizationFailed' exception).
-              - The semantics of which jobs are viewable by
-                the calling user context, or how long a
-                resource manager keeps job information, are
-                implementation dependent.


    - get_job
      Purpose:  Given a job identifier, this method returns a
                job object representing this job.
      Format:   get_job             (in  string job_id,
                                     out job    job)
      Inputs:   job_id:              job identifier as returned
                                     by the resource manager
+     InOuts:   -
      Outputs:  job:                 a job object representing
                                     the job identified by
                                     job_id
+     PreCond:  - job identified by job_id is managed by the
+               job_service.
+     PostCond: -
+     Perms:    Query on the job.
      Throws:   NotImplemented
                BadParameter
                DoesNotExist
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - in general, only a job_service representing the
                  resource manager which submitted the job may be
                  able to handle the job_id, and to identify the
                  job -- however, other job_services may succeed
                  as well.
                - if the resource manager can handle the job_id,
                  but the referenced job is not alive, a
                  'DoesNotExist' exception is thrown.
                - if the resource manager cannot parse the job_id
                  at all, a 'BadParameter' exception is thrown.
```

```
    - get_self
      Purpose:  This method returns a job object representing
                _this_ job, i.e. the calling application.
      Format:   get_self            (out job_self  self)
      Inputs:   -
+     InOuts:   -
      Outputs:  self:                a job_self object
                                     representing _this_ job.
+     PreCond:  - the application is managed by the job_service.
+     PostCond: - job_self is, by definition, in 'Running'
+                 state.
+     Perms:    Query on the job.
      Throws:   NotImplemented
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - in general, only a job_service representing the
                  resource manager which started the application
                  which now calls get_self() can successfully
                  return a job_self instance.  However, other
                  job_services may succeed as well.
                - if a job_service cannot handle the calling job
                  as a job_self instance, a 'NoSuccess' exception
!                 is thrown, with a descriptive error message.
```

Class `job`

The `job` provides the manageability interface to a job instance submitted to a resource manager. There are two general types of methods: those for retrieving job state and information, and those for manipulating the job. The methods intended to manipulate jobs cannot make any guarantees about *how* the resource manager will **affect** an action to be taken. The API implementation is designed to be agnostic of the backend implementation, *such that any backend could be implemented to perform an action*. For example, the checkpoint routine might cause an application level checkpoint, or might use the services of GridCPR.

Job implements the `saga::attributes` interface. If not noted otherwise, none of these attributes is available before the job is running, and none is guaranteed to have a non-empty value while the job is running or after the job finishes.

Job also implements the monitorable interface, and thus allows monitoring and **notification** for changes of **runtime** attributes.

```
-   - CONSTRUCTOR
-     Purpose:  create the object
-     Format:   CONSTRUCTOR       (out job obj);
-     Inputs:   -
-     InOuts:   -
-     Outputs:  obj:               the newly created object
-     PreCond:  -
-     PostCond: - job is in 'Unknown' state.
-     Perms:    -
-     Throws:   NotImplemented
-     Notes:    - the CONSTRUCTOR serves only the purpose to
-                 create jobs to be passed by reference to
-                 the asynchronous create_job method of the
-                 job_service class.
-               - if any method other than the DESTRUCTOR is
-                 called on the created job before it was
-                 initialized by an asynchronous call to
-                 create_job(), an 'IncorrectState' exception
-                 MUST be thrown.
-
-
    - DESTRUCTOR
      Purpose:  destroy the object
      Format:   DESTRUCTOR        (in  job  obj)
      Inputs:   obj:               the object to destroy
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: -
+     Perms:    -
!     Throws:   -
      Notes:    - the object destruction does not imply a
                  call to cancel() for the job instance.


    - get_job_description
      Purpose:  Retrieve the job_description which was used to
                submit this job instance.
      Format:   get_job_description (out job_description jd);
      Inputs:   -
+     InOuts:   -
```

```
      Outputs:  jd:                  a job_description object
+     PreCond:  -
+     PostCond: - jd is deep copied (no state is shared
+                 after method invocation)
+     Perms:    Query
      Throws:   NotImplemented
                DoesNotExist
-               IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - There are cases when the job_description is not
                  available.  This may include cases when
                  the job was not submitted through
                  SAGA and get_job() was used to retrieve the
                  job, or when this state information has been
                  lost (e.g. the client application restarts and
                  the particular SAGA implementation did not
                  persist the information).  In that case, a
!                 'DoesNotExist' exception is thrown, with a
                  descriptive error message.


    - get_stdin
      Purpose:  retrieve input stream for a job.
      Format:   get_stdin          (out opaque stdin)
      Inputs:   -
+     InOuts:   -
      Outputs:  stdin:              standard input stream for
                                    the job
+     PreCond:  - the job is interactive.
+     PostCond: - the jobs standard input stream is available
+                 at stdin.
+     Perms:    Write (application can write to the jobs stdin).
      Throws:   NotImplemented
                BadParameter
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
!     Notes:    - if the preconditions are met, but the standard
```

```
                input stream is not available for some
                reason, a 'DoesNotExist' exception is thrown.
+               - the stream MUST be valid until the job reaches
+                 a final state.  If it is, for some reason,
+                 disconnected earlier, a language typical error
+                 message is thrown (e.g. EBADF could be
+                 returned on writes on that stream in C).
+               - if the job is not interactive, e.g. it was
+                 submitted with the 'Interactive' attribute set
+                 to 'False', an 'IncorrectState' exception is
+                 thrown.
+               - if the job is not in 'New' state, it is not
+                 guaranteed that the job did not receive other
+                 data on its standard input stream before.


    - get_stdout
      Purpose:  retrieve output stream of job
      Format:   get_stdout         (out opaque stdout)
      Inputs:   -
+     InOuts:   -
      Outputs:  stdout:             standard output stream for
                                    the job
+     PreCond:  - the job is interactive.
+     PostCond: - the jobs standard output stream is available
+                 from stdout.
+     Perms:    Read (application can read the jobs stdout).
      Throws:   NotImplemented
                BadParameter
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
!     Notes:    - if the preconditions are met, but the standard
                  output stream is not available for some
                  reason, a 'DoesNotExist' exception is thrown.
+               - the stream MUST be valid until the job reaches
+                 a final state.  If it is, for some reason,
+                 disconnected earlier, a language typical error
+                 message is thrown (e.g. EBADF could be
+                 returned on reads on that stream in C).
+               - if the job is not interactive, e.g. it was
+                 submitted with the 'Interactive' attribute set
```

```
+                 to 'False', an 'IncorrectState' exception is
+                 thrown.
+               - if the job is not in 'New' state, it is not
+                 guaranteed that the job did write data on
+                 its standard output stream before, which are
+                 then not returned on the returned stream.


    - get_stderr
      Purpose:  retrieve error stream of job
      Format:   get_stderr         (out opaque stderr)
      Inputs:   -
+     InOuts:   -
      Outputs:  stderr:             standard error stream for
                                    the job
+     PreCond:  - the job is interactive.
+     PostCond: - the jobs standard error stream is available
+                 from stderr.
+     Perms:    Read (application can read the jobs stderr).
      Throws:   NotImplemented
                BadParameter
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
!     Notes:    - if the preconditions are met, but the standard
                  error stream is not available for some
                  reason, a 'DoesNotExist' exception is thrown.
+               - the stream MUST be valid until the job reaches
+                 a final state.  If it is, for some reason,
+                 disconnected earlier, a language typical error
+                 message is thrown (e.g. EBADF could be
+                 returned on reads on that stream in C).
+               - if the job is not interactive, e.g. it was
+                 submitted with the 'Interactive' attribute set
+                 to 'False', an 'IncorrectState' exception is
+                 thrown.
+               - if the job is not in 'New' state, it is not
+                 guaranteed that the job did write data on
+                 its standard error stream before, which are
+                 then not returned on the returned stream.
```

```
    Job Management Methods:
    -----------------------

    - suspend
      Purpose:  Ask the resource manager to perform a suspend
                operation on the running job.
      Format:   suspend       (void);
      Inputs:   -
+     InOuts:   -
      Outputs:  -
+     PreCond:  - the job is in 'Running' state.
+     PostCond: - the job is in 'Suspended' state.
+     Perms:    Exec (job can be controlled).
      Throws:   NotImplemented
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
!     Notes:    - if the job is not in 'Running' state, an
                  'IncorrectState' exception is thrown.


    - resume
      Purpose:  Ask the resource manager to perform a resume
                operation on a suspended job.
      Format:   resume             (void);
      Inputs:   -
+     InOuts:   -
      Outputs:  -
+     PreCond:  - the job is in 'Suspended' state.
+     PostCond: - the job is in 'Running' state.
+     Perms:    Exec (job can be controlled).
      Throws:   NotImplemented
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
!     Notes:    - if the job is not in 'Suspended' state, an
                  'IncorrectState' exception is thrown.


    - checkpoint
```

```
      Purpose:  Ask the resource manager to initiate a checkpoint
                operation on a running job.
      Format:   checkpoint         (void);
      Inputs:   -
+     InOuts:   -
      Outputs:  -
+     PreCond:  - the job is in 'Running' state.
+     PostCond: - the job is in 'Running' state.
+               - the job was checkpointed.
+     Perms:    Exec (job can be controlled).
      Throws:   NotImplemented
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
!     Notes:    - The semantics of checkpoint(), and the actions
                  taken to initiate a checkpoint, are resource
                  manager specific.  In particular, the
!                 implementation or backend can trigger either
                  a system level or an application level
!               - if the job is not in 'Running' state,
!                 an 'IncorrectState' exception is thrown.


    - migrate
      Purpose:  Ask the resource manager to migrate a job.
      Format:   migrate            (in job_description jd);
      Inputs:   jd:                 new job parameters to apply
                                    when the job is migrated
+     InOuts:   -
      Outputs:  -
+     PreCond:  - the job is in 'Running' or 'Suspended' state.
+     PostCond: - the job keeps its state.
+               - jd is deep copied (no state is shared
+                 after method invocation)
+               - the job reflects the attributes specified in
+                 the job_description.
+     Perms:    Exec (job can be controlled).
      Throws:   NotImplemented
                BadParameter
                IncorrectState
                AuthorizationFailed
                AuthenticationFailed
                PermissionDenied
```

```
                Timeout
                NoSuccess
      Notes:    - jd might indicate new resource
                  requirements, for example.
                - the action of migration might change the job
                  identifier within the resource manager.
                - ideally, the submitted job description was
                  obtained by get_job_description(), and then
!                 changed by the application.  This is not a
-                 condition though.
+                 requirement though.
                - if the job is not in 'Running' or 'Suspended'
!                 state, an 'IncorrectState' exception is thrown.
!               - the method can call the same exceptions as
                  the submit_job() and run() methods, in
                  particular in respect to an incorrect
                  job_description.


    - signal
      Purpose:  Ask the resource manager to deliver an arbitrary
                signal to a dispatched job.
      Format:   signal               (in int signum);
      Inputs:   signum:               signal number to be
                                      delivered
+     InOuts:   -
      Outputs:  -
+     PreCond:  - job is in 'Running' or 'Suspended' state.
+     PostCond: - the signal was delivered to the job.
+     Perms:    Exec (job can be controlled).
      Throws:   NotImplemented
                BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - there is no guarantee that the signal number
                  specified is valid for the operating system
                  on the execution host where the job is
                  running, or that the signal can be delivered.
!               - if the signal number is not supported by the
                  backend, a 'BadParameter' exception is thrown.
                - if the job is not in 'Running' or 'Suspended'
!                 state, an 'IncorrectState' exception is
```

```
                  thrown.
```

Class `job_self`

The `job_self` class `IS-A` job which represents the current application (i.e. the very application which owns that `job_self` instance). It can only by created by calling `get_self()` on a job service (that call can fail though).

The motivation to introduce this class is twofold: (1) it allows to actively handle the current application as a grid job (e.g. to migrate it, or to obtain its job description for cloning/spawning); (2) as the class implements the steerable interface, it is possible to add `ReadWrite` metrics to its instance – that way it is possible to expose these metrics to other external applications, which in fact allows to steer the current application.

A drawback of this approach is that, in order to make an application steerable, a `job_service` instance is needed which can in fact return a `job_self` instance, which means there must be a resource manager available which can manage the current application – that however has nothing to do with the concept of remote steering. Future versions of the SAGA API may change that, and may make `job_self` a singleton, independent from the `job_service` behaviour. As a result, that class might disappear, and might not be maintained for backward compatibility.

```
-   - CONSTRUCTOR
-     Purpose:  create the object
-     Format:   CONSTRUCTOR       (out job_self self);
-     Inputs:   -
-     InOuts:   -
-     Outputs:  self:              the newly created object
-     PreCond:  -
-     PostCond: - self is in 'Unknown' state
-     Perms:    -
-     Throws:   NotImplemented
-     Notes:    - the constructor serves only the purpose to
-                 create jobs to be passed by reference to
-                 the asynchronous get_self method of the
-                 job_service class.
-               - if any method other then the DESTRUCTOR is
-                 called on the created job_self before it was
-                 initialized by an asynchronous call to
-                 get_self(), an 'IncorrectState' exception
```

```
-                 MUST be thrown.
-
-
    - DESTRUCTOR
      Purpose:  destroy the object
      Format:   DESTRUCTOR        (in  job_self  obj)
      Inputs:   obj:               the object to destroy
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   -
      Notes:    - the object destruction does not imply a
                  call to cancel() for the job_self instance.
```

4.1.8 Examples

Code Example

```
1   Example : simple job submission and polling for finish.
2
3   // ------------------------------------------------
4   // c++ example
5   std::list <std::string>    transfers;
6   saga::job_description jobdef;
7   saga::job_service     js;
8
9   transfers.push_back ("infile > infile");
10  transfers.push_back ("ftp://host.net/path/out << outfile");
11
12  jobdef.set_attribute        ("CandidateHost", "hostname");
13  jobdef.set_attribute        ("Executable",    "job.sh");
14  jobdef.set_attribute        ("TotalCPUCount", "16");
15  jobdef.set_vector_attribute ("FileTransfer",  transfers);
16
17  saga::job job = js.create_job (jobdef);
18
19  job.run ();
20
21  while ( 1 )
22  {
23    // get job state
24    saga::job::state state = job.get_state ();
25
26    // get list of hosts the job is/where running on
27    std::list <std::string> hostlist = job.get_attribute
```

```
                                        ("ExecutionHosts");

  if ( saga::job::Running == state )
  {
    std::cout << "Job is running." << std::endl;
  }
  else if ( saga::job::Suspended == state )
  {
    std::cout << "Job is suspended." << std::endl;
  }
  else if ( saga::job::Done == state )
  {
    std::cout << "Job completed successfully." << std::endl;
    exit (0);
  }
  else if ( saga::job::Canceled == state )
  {
    // this should never occur, as cancel is not called.
    std::cout << "Job canceled." << std::endl;
    exit (1);
  }
  else
  {
    // state can only be 'Failed'
    assert (saga::job::Failed == state);

    std::string exitcode = job.get_attribute ("ExitCode");

    std::cout << "Job failed with exitcode:"
              << exitcode
              << std::endl;
    exit ( atoi(exitcode) );
  }

  sleep (1); // idle
}
```

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 these classes which operate on arbitrary hierarchical name spaces, such as used in physical, virtual, and logical file systems, and in information systems.

The API is inspired by the POSIX standard, which defines tools and calls to handle the name space of physical files *and* directories. The methods listed for the interfaces have POSIX-like syntax and semantics.

While POSIX has an iterative interface to directory listing (i.e. opendir, telldir, seekdir, readdir), the corresponding part of the interface included here deviates significantly from the POSIX version: it has fewer calls, with a different syntax, but identical semantics.

Please note that 'stat'-like API calls are *not* covered here – they are rather meaningless on a name space per se, but belong to the specific implementations, e.g. physical files, which *inherit* the `namespace` *classes*.

4.2.1 Definitions

The Grid File System Working Group in OGF has defined a Resource Namespace Service (RNS [20]). The SAGA Core API specification follows the definition of a name space from that document.

Directory: A 'Directory' represents what [20] defines as 'Virtual Directory':

> *"A virtual directory is an RNS entry that is represented as a non-leaf node in the hierarchical name space tree. When rendered by a name space service client, a virtual directory functions similar to that of a standard filesystem directory or registry key. It is considered virtual because it does not have any corresponding representation outside of the name space. A virtual directory, therefore, is purely a name space entity that functions in much the same way as a conventional filesystem directory or registry key by maintaining a list of subentries, which thereby demonstrate a hierarchical relationship. There are no restrictions regarding the layout of the name space tree; both virtual directories and junctions can be nested within nested virtual directories recursively.*
>
> *A virtual directory may be considered analogous to a collection, category, or context – to the extent that these terms are used in most directory, registry, or catalogue contexts. Virtual directories do not have any time or space existence outside of the name space and strictly serve to facilitate hierarchy. Name space hierarchies offer categorization or grouping*

of entries, by presenting the illusion of compartments, which may contain sub-compartments as well as junctions."

Directory Entry: A *directory entry* or *entry* represent what [20] defines as 'Junction'. Note that any type of junction defined there could be used:

> *"A junction is an RNS entry that interconnects a reference to an existing resource into the hierarchical name space. Junctions represent a name-to-resource mapping that is composed of a human oriented index key or 'name' that maps to an endpoint reference. The endpoint reference may refer to any addressable resource, which includes other name space entries, as well as names or unique identifiers to be resolved by other resolution service, as well as definitive target consumable resource. All compliant RNS implementations MUST embody the target information of a name space junction within a valid WS-Addressing [...] Endpoint Reference (EPR)."*

Pathnames: A *pathname* as accepted by this specification MUST be either formatted as URLs or MUST follow the specification of entry names as described *in section 1.2.2.1 "Entry Name Restrictions" in [20]* *in [20], Section 1.2.2.1 "Entry Name Restrictions"* (formatting changed):

> *"Entry names are composed of a simple string of human readable characters. Since certain characters serve special purposes both within the name space service and within a number of systems that may use this service, this section describes the mandatory restrictions for all entry names:*
>
> *Names MUST NOT...*
>
> - *Contain any of the following characters:* `/ : ; * ? " < > |`
> - *Contain any non-readable characters, such as the carriage return (ANSI 13) or line feed (ANSI 10) or tab (ANSI 9)*
> - *Be greater than 255 characters in length (Unicode)*
>
> *Names SHOULD...*
>
> - *Accommodate Unicode characters*
> - *Be easily readable by a human user, suggesting less than 32 characters per name*
>
> *Names MAY...*
>
> - *Contain space (ANSI 32) characters*

> *Notice these restrictions apply to entry names and are not describing paths. Paths are constructed of one or more entry names separated by the forward slash character (/)".*

Note that, in fact, pathnames as specified above are syntactically valid URLs, and this specification is therefore only refering to URLs. Both, SAGA implementations and SAGA usage SHOULD, however, strive for compliance with [20]. An exception is the use of relative pathnames which, in SAGA, can contain wildcards (see below).

If pathnames are specified as URLs, the path section of the URL MUST follow the guidelines for pathnames as cited above, and MUST NOT contain any parameter, query or fragment parts.

Additionally, pathname specifications in SAGA can contain wildcards as specified below.

All method arguments which are named `name`, `source` or `target` are considered pathnames. These pathnames can always be relative pathnames (i.e. they can be relative to the *current working directory* (`cwd`) of the object instance the operation is performed upon, e.g. when they start with `'./'` or `'../'`).

Note that the comments from Section 2.11, apply here. In particular, an implementation MAY throw an `IncorrectURL` exception if it is unable to handle a given *URL , e.g. because of its scheme*.

Current Working Directory (cwd) Every `saga::ns_entry` instance has an associate current working directory (cwd), which forms the implicit base for all operations on relative pathnames. For `saga::ns_directory` instances, that cwd can be changed with the `change_dir` method. Otherwise, cwd only changes if the entry itself is `move()`'d.

Links: *Links* in this specification are considered *symbolic links*, i.e. they can break if the entry they point to is removed. An implementation MAY support links, as not all backends can support links, and others might support links only in specific circumstances (e.g. if entry and link live on the same file system).

The 'Dereference' flag allows methods to operate on the link target instead of the link – only one level of reference is resolved though. The `read_link()` method does also resolve only one link level, and returns a URL pointing to the link target.

At the moment, [20] does not have a notion of symbolic links. However, an RNS 'junction' which is associated with another RNS junction can be regarded as *a* symbolic link.

Wildcards: The API supports wildcards *where appropriate for a number of calls, as listed below*, and thereby follows the POSIX standard [21, 22, 23] for shell wildcards. Available wildcard patterns are:

`*`	: matches any string
`?`	: matches a single character
`[abc]`	: matches any of a set of characters
`[a-z]`	: matches any of a range of characters
`[!abc]`	: matches none of a range of characters
`[!a-z]`	: matches none of a range of characters
`{a,bc}`	: matches any of a set of strings

See the POSIX standard [21, 22, 23] for more details. In the *SAGA* API, wildcards are allowed in all pathnames where they can be used in the respective shell commands, as:

```
copy   *.txt dir
move   *.txt dir
link   *.txt dir
ls     *.txt
remove *.txt
```

Users are rarely aware that wildcards can be used in unorthodox places, such as:

`move *.txt dir*`

`move *`

The result of such operations is dependend on the order the wildcard expansion is performed, e.g. if 'dir' expands to 'dir_1 dir_2', all txt files and dir_1 will end up in dir_2. SAGA implementations MUST support wildcards for all pathnames where that ambiguity cannot arise. (source for move etc.), and MAY support wildcards at all pathnames where that ambiguite may arise.*

For the method calls on `saga::ns_entry`*, no wildcards are allowed.Note that the methods* `read_link()`, `exists()`, `is_dir()`, `is_entry()`, `is_link()`, `open()`, *and* `open_dir()` *MUST NOT support wild cards (their return values make only sense in respect to a single entry). Note that only those methods MUST support wildcards for which this is explicitly specified here. Other methods MUST NOT support wildcards, as this would not be meaningful.* Flags MUST be applied to all elements of a wildcard expansion, even if that raises an exception for any reason.

For the use of wildcards, separate calls are provided which accept strings instead of URLs. The reason for this is that RFC 3986 [5], which defines the syntax of URLs, explicitly forbids most POSIX wildcard characters as part of a URL. Also, we feel that wildcards make most sense in relative pathnames (i.e. relative to a working directory). Strings in these separate calls thus MUST be relative paths,

and thus MUST only contain URL path elements, whereby the path element MUST NOT start with an '/'. Apart from that, the semantics of the wildcard-enabled string method versions of the calls are identical to the semantics of their respective URL counterparts. If the method encounters any error condition on any one of the expanded URLs, an exception is thrown, and the state of the other (valid or invalid) expanded URL targets remains undefined.

a long section about ACLs got removed

Opening and Closing Name Space Entries: If a `ns_entry` object instance gets created, it is also opened. Hence, the semantics and all notes of the respective `open()` call also apply to the constructor. The same holds for all classes that inherit `ns_entry`.

In accordance with Section 2.5.4, the `saga::ns_entry` class has a `close()` method, which allows to enforce a timely release of used (local and remote) resources. After a `ns_entry` instance was closed, all method calls on that instance (apart from the `DESTRUCTOR`) MUST throw an `IncorrectState` exception. A destruction of an entry implies the respective `close()` semantics. The same holds for all classes that inherit `ns_entry`.

If an entry gets successfully opened without specifying 'Lock' as open flag, its state may get currupted if some other backend operation removes or moves the opened entity, or changes its state. In that case, any subsequent operation on the object instance can fail unexpectedly. An `IncorrectState` exception describing the type of state change SHOULD be thrown if such a state change is detected and causes an operation to fail. Otherwise, the normal exception indicating the type of error which occured *SHOULD* be thrown. The `IncorrectState` exception is thus listed on most method calls below, but not individually motivated unless it is also used in any other semantic context.

4.2.2 Specification

```
- package saga.name_space
+ package saga.namespace
  {
    enum flags
    {
      None            =   0,
      Overwrite       =   1,
      Recursive       =   2,
```

```
      Dereference     =   4,
      Create          =   8,
!     Exclusive       =  16,
      Lock            =  32,
      CreateParents   =  64,
    }

-
-   enum acl
-   {
-     None      =  0,
-     ACL_List  =  1,
-     ACL_Read  =  2,
-     ACL_Write =  4,
-     ACL_Exec  =  8,
-     ACL_Admin = 16
-   }
-

    class ns_entry : implements   saga::object,
                     implements   saga::async
+                    implements   saga::permissions
                  // from object  saga::error_handler
    {
!     CONSTRUCTOR        (in  session        s,
                         in  saga::url      name,
                         in  int            flags   = None);
                         out ns_entry       obj     );
      DESTRUCTOR         (in  ns_entry       obj     );

      // basic properties
      get_url            (out saga::url      url     );
      get_cwd            (out saga::url      cwd     );
      get_name           (out saga::url      name    );

      // navigation/query methods
-     is_dir             (in  int            flags   = None,
-                         out boolean        test    );
+     is_dir             (out boolean        test    );
-     is_entry           (in  int            flags   = None,
-                         out boolean        test    );
+     is_entry           (out boolean        test    );
-     is_link            (in  int            flags   = None,
-                         out boolean        test    );
+     is_link            (out boolean        test    );
      read_link          (out saga::url      link    );
```

```
-     // security
-     set_acl             (in  string         dn,
-                          in  int            acl,
-                          in  int            flags = None);
-     get_acl             (in  int            flags = None,
-                          out int            acl     );
-     list_dn             (in  int            flags = None,
-                          out array<string>  dn      );
-
      // management methods
      copy                (in  saga::url      target,
                           in  int            flags = None);
      link                (in  saga::url      target,
                           in  int            flags = None);
      move                (in  saga::url      target,
                           in  int            flags = None);
      remove              (in  int            flags = None);
      close               (in  float          timeout = 0.0);
+
+
+     // permissions with flags
+     permissions_allow   (in  string         id,
+                          in  permission     perm,
+                          in  int            flags = None);
+     permissions_deny    (in  string         id,
+                          in  permission     perm,
+                          in  int            flags = None);
    }


    class ns_directory : extends        saga::ns_entry
                         // from ns_entry  saga::object
                         // from ns_entry  saga::async
+                        // from ns_entry  saga::permissions
                         // from object    saga::error_handler
    {
!     CONSTRUCTOR         (in  session        s,
                          in  saga::url      name,
                          in  int            flags = None,
                          out ns_directory   obj     );
      DESTRUCTOR          (in  ns_directory   obj     );

      // navigation/query methods
      change_dir          (in  saga::url      dir     );
!     list                (in  string         name_pattern = ".",
```

```
                        in  int            flags   = None,
                        out array<saga::url> names   );
      find              (in  string         name_pattern,
                        in  int            flags = Recursive,
                        out array<saga::url> names   );
      read_link         (in  saga::url      name,
                        out saga::url      link    );
      exists            (in  saga::url      name,
                        out boolean        exists  );
      is_dir            (in  saga::url      name,
-                       in  int            flags = None,
                        out boolean        test    );
      is_entry          (in  saga::url      name,
-                       in  int            flags = None,
                        out boolean        test    );
      is_link           (in  saga::url      name,
-                       in  int            flags = None,
                        out boolean        test    );

      // manage entries by number
      get_num_entries   (out int            num     );
      get_entry         (in  int            entry,
                        out saga::url      name    );

-     // security
-     set_acl           (in  saga::url      name,
-                       in  string         dn,
-                       in  int            acl,
-                       in  int            flags = None);
-     get_acl           (in  saga::url      name,
-                       in  int            flags = None,
-                       out int            acl     );
-     list_dn           (in  saga::url      name,
-                       in  int            flags = None,
-                       out array<string>  dn      );
-
      // management methods
      copy              (in  saga::url      source,
                        in  saga::url      target,
                        in  int            flags = None);
      link              (in  saga::url      source,
                        in  saga::url      target,
                        in  int            flags = None);
      move              (in  saga::url      source,
                        in  saga::url      target,
                        in  int            flags = None);
```

```
      remove             (in  saga::url      target,
                          in  int            flags = None);
      make_dir           (in  saga::url      target,
                          in  int            flags = None);

+     // management methods - wildcard versions
+     copy               (in  string         source,
+                         in  saga::url      target,
+                         in  int            flags = None);
+     link               (in  string         source,
+                         in  saga::url      target,
+                         in  int            flags = None);
+     move               (in  string         source,
+                         in  saga::url      target,
+                         in  int            flags = None);
+     remove             (in  string         target,
+                         in  int            flags = None);

      // factory methods
      open               (in  saga::url      name,
                          in  int            flags = None,
                          out ns_entry       entry   );
      open_dir            (in  saga::url     name,
                          in  int            flags = None,
                          out ns_directory   dir     );
+
+
+     // permissions with flags
+     permissions_allow  (in  saga::url      target,
+                         in  string         id,
+                         in  int            perm,
+                         in  int            flags = None);
+     permissions_deny   (in  saga::url      target,
+                         in  string         id,
+                         in  int            perm,
+                         in  int            flags = None);
+
+     // permissions with flags - wildcard versions
+     permissions_allow  (in  string         target,
+                         in  string         id,
+                         in  int            perm,
+                         in  int            flags = None);
+     permissions_deny   (in  string         target,
+                         in  string         id,
+                         in  int            perm,
+                         in  int            flags = None);
```

```
    }
  }
```

4.2.3 Specification Details

Enum `flags`

The ***`flags`*** *describe the properties of several operations on namespace entries. Packages which inherit from the namespace package use the same flag semantics unless specified otherwise, but will, in general, add additional flags to some operations.*

`None`
: *indicates the absence of flags, and thus also implies that the default flags for an operation do not apply, either.*

`Overwrite`
: *enforces an operation which creates a new namespace entry to continue even if the target entry does already exist – if that flag is not given, an* ***`'AlreadyExists'`*** *exception would result from such an operation.*

`Recursive`
: *enforces an operation to apply recursively on a directory tree – if that flag is not given, the same operation would only apply to the given directory, and not to its children.*

`Dereference`
: *enforces an operation to apply not to the entry pointed to by the target name, but to the link target of that entry – if that flag is not given, the same operation would apply to the entry directly, and its link target stays unaffected.*

`Create`
: *allows a namespace entry to be created while opening it, if it does not alreay exist – if that flag is not given, the same open operation would cause a* ***`'DoesNotExist'`*** *exception. If the entry exists, the flag is ignored.*

`Exclusive`
: *implies a modification to the meaning of the* ***`Create`*** *flag: if the entry already exists, the* ***`Create`*** *flag is is no longer silently ignored, but causes an* ***`'AlreadyExists'`*** *exception.*

`Lock`
: *enforces a lock on the name space entry when it is opened. Locks are advisory in SAGA, semantic details for locking are defined in the description of the* ***`open()`*** *call.*

`CreateParents`

An operation which would create a name space entry would normally fail if any path element in the targets name does not yet exist. If this flag is given, such an operation would not fail, but would imply that the missing path elements are created on the fly.

Class `ns_entry`

`ns_entry` defines methods which serve the inspection of the entry itself, methods which allows to manage the entry (e.g. to copy, move, or remove it), and methods to manipulate the **entry's** access control lists.

In general, multiple such URLs might be valid to identify an entry:

```
 ftp://ftp.host.net/pub/data/test.txt
http://www.host.net/ftp/data/test.txt
http://www.host.net/ftp/data/./test.txt
http://www.host.net/ftp/data/../data/test.txt
```

Any valid URL can be returned on `get_url()`, but it SHOULD not contain `'..'` or `'.'` path elements, *i.e. should have a normalized path element*. The URL returned on `get_url()` should serve as base for the return values on `get_cwd()` and `get_name()`: *In general it should hold:*

```
get_url() = get_cwd() + '/' + get_name()
```

```
-   Constructor / Destructor:
-
    - CONSTRUCTOR
      Purpose:  create the object
!     Format:   CONSTRUCTOR          (in  session   s,
                                      in  saga::url name,
                                      in  int       flags = None,
                                      out ns_entry obj)
!     Inputs:   s:                   session handle
                name:                initial working dir
                flags:               open mode
+     InOuts:   -
      Outputs:  obj:                 the newly created object
+     PreCond:  -
+     PostCond: - the entry is opened.
+               - 'Owner' of target is the id of the context
```

```
+                 use to perform the opereration, if the
+                 entry gets created.
+     Perms:      Exec  for parent directory.
+                 Write for parent directory if Create is set.
+                 Write for name if Write is set.
+                 Read  for name if Read  is set.
      Throws:     NotImplemented
                  IncorrectURL
                  BadParameter
                  DoesNotExist
                  AlreadyExists
                  PermissionDenied
                  AuthorizationFailed
                  AuthenticationFailed
                  Timeout
                  NoSuccess
!     Notes:      - the default flags are 'None' (0)
                  - the constructor performs an open of the
                    entry - all notes to the respective open
                    call (on namespace_directory) apply.


    - DESTRUCTOR
      Purpose:  destroy the object
      Format:   DESTRUCTOR           (in  ns_entry     obj)
      Inputs:   obj:                  the object to destroy
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - the entry is closed.
+     Perms:    -
      Throws:   -
      Notes:    - if the instance was not closed before, the
!                 destructor performs a close() on the instance,
                  and all notes to close() apply.



    Methods for inspecting ns_entry:
    --------------------------------

    - get_url
      Purpose:  obtain the complete url pointing to the entry
      Format:   get_url            (out  saga::url url);
      Inputs:   -
+     InOuts:   -
```

```
      Outputs:  url                 url pointing to the entry
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   NotImplemented
                IncorrectState
                Timeout
                NoSuccess
+     Notes:    -
-     Notes:    - if the instance was not opened before, an
-                 'IncorrectState' exception is thrown.



    - get_cwd
      Purpose:  obtain the current working directory for the
                entry
      Format:   get_cwd            (out saga::url cwd);
      Inputs:   -
+     InOuts:   -
      Outputs:  cwd                 current working directory
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   NotImplemented
                IncorrectState
                Timeout
                NoSuccess
+     Notes:    -
-     Notes:    - if the instance was not opened before, an
-                 'IncorrectState' exception is thrown.
+               - returns the directory part of the url path
+                 element.



    - get_name
      Purpose:  obtain the name part of the url path element
      Format:   get_name           (out  saga::url name);
      Inputs:   -
+     InOuts:   -
      Outputs:  name                last part of path element
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   NotImplemented
```

```
                IncorrectState
                Timeout
                NoSuccess
+     Notes:    -
-     Notes:    - if the instance was not opened before, an
-                 'IncorrectState' exception is thrown.



    - is_dir
!     Purpose:  tests the entry for being a directory
-     Format:   is_dir             (in  int     flags = None,
+     Format:   is_dir             (out boolean test);
!     Inputs:   -
+     InOuts:   -
      Outputs:  test:               boolean indicating if entry
                                    is a directory
+     PreCond:  -
+     PostCond: -
+     Perms:    Query
+               Query for parent directory.
      Throws:   NotImplemented
-               BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - returns true if entry is a directory, false
                  otherwise
-               - flags can be set to 'Dereference', the default
-                 is 'None' (0).
-               - other flags are not allowed on this method,
-                 and cause a 'BadParameter' exception.
-               - if the instance was not opened before, an
-                 'IncorrectState' exception is thrown.
                - similar to 'test -d' as defined by POSIX.



    - is_entry
!     Purpose:  tests the entry for being an ns_entry
-     Format:   is_entry           (in  int     flags = None,
+     Format:   is_entry           (out boolean test);
!     Inputs:   -
+     InOuts:   -
```

```
      Outputs:  test:               boolean indicating if entry
!                                   is an ns_entry
+     PreCond:  -
+     PostCond: -
+     Perms:    Query
+               Query for parent directory.
      Throws:   NotImplemented
-               BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - the method returns false if the entry is a
!                 link or a directory (although an ns_directory
                  IS_A ns_entry, false is returned on a test on
!                 an ns_directory) - otherwise true is returned.
-               - flags can be set to 'Dereference', the default
-                 is 'None' (0)
-               - other flags are not allowed on this method,
-                 and cause a 'BadParameter' exception.
-               - if the instance was not opened before, an
-                 'IncorrectState' exception is thrown.
!               - similar to 'test -f' as defined by POSIX.


    - is_link
!     Purpose:  tests the entry for being a link
-     Format:   is_link            (in  int     flags = None,
+     Format:   is_link            (out boolean test);
!     Inputs:   -
+     InOuts:   -
      Outputs:  test:               boolean indicating if
                                    entry is a link
+     PreCond:  -
+     PostCond: -
+     Perms:    Query
+               Query for parent directory.
      Throws:   NotImplemented
-               BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
```

```
                NoSuccess
      Notes:    - returns true if the entry is a link, false
                  otherwise
-               - flags can be set to 'Dereference', the default
-                 is 'None' (0)
-               - other flags are not allowed on this method,
-                 and cause a 'BadParameter' exception.
-               - if the instance was not opened before, an
-                 'IncorrectState' exception is thrown.
!               - similar to 'test -L' as defined by POSIX.


    - read_link
      Purpose:  returns the name of the link target
      Format:   read_link          (out saga::url link);
      Inputs:   -
+     InOuts:   -
      Outputs:  link:               resolved name
+     PreCond:  -
+     PostCond: -
+     Perms:    Query
+               Query for parent directory.
      Throws:   NotImplemented
-               BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - the returned name MUST be sufficient to
                  access the link target entry
                - resolves one link level only
                - if the entry instance this method is called
!                 upon does not point to a link, an
!                 'IncorrectState' exception is thrown.
-               - if the instance was not opened before, an
-                 'IncorrectState' exception is thrown.
                - similar to 'ls -L' as defined by POSIX.

-
-
-   Methods for managing access control lists:
-   ------------------------------------------
-
-   - set_acl
```

```
-     Purpose:  set access control list for this entry
-     Format:   set_acl            (in  string  dn,
-                                   in  int     acl,
-                                   in  int     flags = None);
-     Inputs:   dn:                 DN to set ACLs for
-               acl:                OR'ed ACLs to be set on the
-                                   entity, for the specified dn
-               flags:              flags defining the operation
-                                   modus
-     Outputs:  -
-     Perms:    -
-     Throws:   NotImplemented
-               BadParameter
-               IncorrectState
-               PermissionDenied
-               AuthorizationFailed
-               AuthenticationFailed
-               Timeout
-               NoSuccess
-     Notes:    - if the entry is a directory and the 'Recursive'
-                 flag is set, the ACLs are applied to all
-                 entries in the directory tree below.  If the
-                 flag is set and the entry is not a directory, a
-                 'BadParameter' exception is thrown.
-               - if the entry is a link and the 'Dereference'
-                 flag is set, the ACLs are set for the link
-                 target, and not for the link itself.  If the
-                 flag is set and the entry is not a link, a
-                 'BadParameter' exception is thrown.
-               - other flags are not allowed, and cause a
-                 'BadParameter' exception.
-               - the default flags are 'None' (0).
-               - invalid or inconsistent acl specifications
-                 cause a 'BadParameter' exception with a
-                 descriptive error message
-               - if the 'dn' cannot be parsed or evaluated, or
-                 if some wildcard in the 'dn' is not supported,
-                 a 'BadParameter' exception with a descriptive
-                 error message is thrown.
-               - if the instance was not opened before, an
-                 'IncorrectState' exception is thrown.
-
-
-   - get_acl
-     Purpose:  get access control list for this entry
-     Format:   get_acl            (in  string   dn,
```

```
-                                   in  int      flags = None,
-                                   out int      acl);
-     Inputs:   dn:                 DN to get ACLs for
-               flags:              flags defining the operation
-                                   modus
-     Outputs:  acl:                OR'ed ACLs set on the
-                                   entity, for the specified dn
-     Perms:    -
-     Throws:   NotImplemented
-               BadParameter
-               IncorrectState
-               PermissionDenied
-               AuthorizationFailed
-               AuthenticationFailed
-               Timeout
-               NoSuccess
-     Notes:    - if the entry is a link and the 'Dereference'
-                 flag is set, the ACLs are retrieved for the
-                 link target, and not for the link itself.
-                 If the flag is set and the entry is not a
-                 link, a 'BadParameter' exception is thrown.
-               - other flags are not allowed, and cause a
-                 'BadParameter' exception.
-               - the default flags are 'None' (0).
-               - invalid or inconsistent acl specifications
-                 cause a 'BadParameter' exception with a
-                 descriptive error message
-               - if the 'dn' cannot be parsed or evaluated, or
-                 if some wildcard in the 'dn' is not supported,
-                 a 'BadParameter' exception with a descriptive
-                 error message is thrown.
-               - if the instance was not opened before, an
-                 'IncorrectState' exception is thrown.
-
-
-   - list_dn
-     Purpose:  list all DN's for which ACLs are set.
-     Format:   list_dn            (in  int    flags = None,
-                                   out array<string>  dn);
-     Inputs:   flags:              flags defining the operation
-     Outputs:  dn:                 list of DNs for which ACLs
-                                   are set on the entry
-     Perms:    -
-     Throws:   NotImplemented
-               BadParameter
-               IncorrectState
```

```
-               PermissionDenied
-               AuthorizationFailed
-               AuthenticationFailed
-               Timeout
-               NoSuccess
-     Notes:    - if the entry is a link and the 'Dereference'
-                 flag is set, the DNs are retrieved for the
-                 link target, and not for the link itself.
-                 If the flag is set and the entry is not a
-                 link, a 'BadParameter' exception is thrown.
-               - other flags are not allowed, and cause a
-                 'BadParameter' exception.
-               - the default flags are 'None' (0).
-               - the list of returned DNs can contain wildcards
-                 as described earlier.  These can be expanded
-                 by the application if that is required, or can
-                 be reused as they are.
-               - if the instance was not opened before, an
-                 'IncorrectState' exception is thrown.
-

    Methods for managing the name space entry:
    ------------------------------------------

    - copy
      Purpose:  copy the entry to another part of the name space
      Format:   copy               (in  saga::url target,
                                    in  int       flags = None);
      Inputs:   target:             name to copy to
                flags:              flags defining the operation
                                    modus
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - an identical copy exists at target.
+               - 'Owner' of target is the id of the context
+                 use to perform the opereration, if target gets
+                 created.
+     Perms:    Query
+               Exec  for parent directory.
+               Query for target.
+               Query for target's parent directory.
+               Exec  for target's parent directory.
+               Write for target
+                     if  target does exist.
+               Write for target's parent directory
```

```
+                    if  target does not exist.
      Throws:   NotImplemented
+               IncorrectURL
                BadParameter
                DoesNotExist
                AlreadyExists
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - if the target is a directory, the source entry
                  is copied into that directory
!               - a 'BadParameter' exception is thrown if the
                  source is a directory and the 'Recursive' flag
                  is not set.
!               - a 'BadParameter' exception is thrown if the
                  source is not a directory and the 'Recursive'
                  flag is set.
                - if the target lies in a non-existing part of
!                 the name space, a 'DoesNotExist' exception is
                  thrown, unless the 'CreateParents' flag is
                  given - then that part of the name space must
                  be created.
                - if the target already exists, it will be
                  overwritten if the 'Overwrite' flag is set,
!                 otherwise it is an 'AlreadyExists' exception.
+               - if a directory is to be copied recursively,
+                 but the target exists and is not a directory,
+                 and not a link to a directory, an
+                 'AlreadyExists' exception is thrown even if
+                 the 'Overwrite' flag is set.
                - if the instance points at an symbolic link,
                  the source is deeply dereferenced before copy.
                  If derefencing is impossible (e.g. on a broken
                  link), an 'IncorrectState' exception is thrown.
                - other flags are not allowed, and cause a
!                 'BadParameter' exception.
!               - the default flags are 'None' (0).
                - similar to 'cp' as defined by POSIX.


    - link
!     Purpose:  create a symbolic link from the target entry to
!               the source entry ( this entry) so that any reference
```

```
!               to the target refers to the source entry
      Format:   link               (in  saga::url target,
                                    in  int       flags = None);
      Inputs:   target:             name to link to
                flags:              flags defining the operation
                                    modus
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - a symbolic link to the entry exists at target.
+               - 'Owner' of target is the id of the context
+                 use to perform the opereration if target gets
+                 created.
+     Perms:    Query
+               Exec  for parent directory.
+               Query for target.
+               Query for target's parent directory.
+               Exec  for target's parent directory.
+               Write for target
+                     if  target does exist.
+               Write for target's parent directory
+                     if  target does not exist.
      Throws:   NotImplemented
+               IncorrectURL
                BadParameter
                DoesNotExist
                AlreadyExists
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - if the target is a directory, the source entry
                  is linked into that directory
                - if the source is a directory, and the
                  'Recursive' flag is set, the source directory
                  is recursively linked to the target (which must
                  be a directory as well - otherwise a
                  'BadParameter' exception is thrown).  The
                  method then behaves similar to lndir.  If the
                  'Recursive' flag is not set, the source entry
                  itself is linked.
!               - a 'BadParameter' exception is thrown if the
                  source is not a directory and the 'Recursive'
                  flag is set.
```

```
                - if the target lies in a non-existing part of
!                 the name space, a 'DoesNotExist' exception is
                  thrown, unless the 'CreateParents' flag is
                  given - then that part of the name space must
                  be created.
                - if the target already exists, it will be
                  overwritten if the 'Overwrite' flag is set,
!                 otherwise it is an 'AlreadyExists' exception.
+               - if a directory is to be moved, but the target
+                 exists and is not a directory, and not a link
+                 to a directory, an 'AlreadyExists' exception
+                 is thrown even if the 'Overwrite' flag is set.
                - if the instance points at an symbolic link,
                  the source is not dereferenced before linking,
                  unless the 'Dereference' flag is given.  If
                  derefencing is impossible (e.g. on a broken
                  link), an 'IncorrectState' exception is thrown.
                - other flags are not allowed, and cause a
!                 'BadParameter' exception.
!               - the default flags are 'None' (0).
                - similar to 'ln' as defined by POSIX.


    - move
      Purpose:  rename source to target, or move source to
!               target if target is a directory.
      Format:   move               (in  saga::url target,
                                    in  int       flags = None);
      Inputs:   target:             name to move to
                flags:              flags defining the operation
                                    modus
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - an identical copy exists at target.
+               - the original entry is removed.
+               - 'Owner' of target is the id of the context
+                 use to perform the opereration if target gets
+                 created.
+     Perms:    Query
+               Write
+               Exec  for parent directory.
+               Write for parent directory.
+               Query for target.
+               Exec  for target's parent directory.
+               Write for target
```

```
+                     if  target does exist.
+                Write for target's parent directory
+                     if  target does not exist.
       Throws:   NotImplemented
+                IncorrectURL
                 BadParameter
                 DoesNotExist
                 AlreadyExists
                 IncorrectState
                 PermissionDenied
                 AuthorizationFailed
                 AuthenticationFailed
                 Timeout
                 NoSuccess
       Notes:    - if the target is a directory, the source entry
                   is moved into that directory
!                - a 'BadParameter' exception is thrown if the
                   source is a directory and the 'Recursive' flag
                   is not set.
!                - a 'BadParameter' exception is thrown if the
                   source is not a directory and the 'Recursive'
                   flag is set.
                 - if the target lies in a non-existing part of
!                  the name space, a 'DoesNotExist' exception is
                   thrown, unless the 'CreateParents' flag is
                   given - then that part of the name space must
                   be created.
                 - if the target already exists, it will be
                   overwritten if the 'Overwrite' flag is set,
!                  otherwise it is an 'AlreadyExists' exception.
                 - if the instance points at an symbolic link,
                   the source is not dereferenced before moving,
                   unless the 'Dereference' flag is given.
                   If derefencing is impossible (e.g. on a broken
                   link), an 'IncorrectState' exception is thrown.
                 - other flags are not allowed, and cause a
!                  'BadParameter' exception.
!                - the default flags are 'None' (0).
                 - similar to 'mv' as defined by POSIX.


    - remove
      Purpose:  removes this entry, and closes it
      Format:   remove             (in  int  flags = None);
      Inputs:   target:             entry to be removed
+     InOuts:   -
```

```
      Outputs:  -
+     PreCond:  -
+     PostCond: - the original entry is closed and removed.
+     Perms:    Query
+               Write
+               Exec  for parent directory.
+               Write for parent directory.
      Throws:   NotImplemented
                BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
!     Notes:    - a 'BadParameter' exception is thrown if the
                  source is a directory and the 'Recursive' flag
                  is not set.
!               - a 'BadParameter' exception is thrown if the
                  source is not a directory and the 'Recursive'
                  flag is set.
                - the source will not be dereferenced unless the
                  'Dereference' flag is given.  If derefencing is
                  impossible (e.g. on a broken link), an
                  'IncorrectState' exception is thrown.
                - other flags are not allowed, and cause a
!                 'BadParameter' exception.
!               - the default flags are 'None' (0).
!               - if the instance was not closed before, this
!                 call performs a close() on the instance, and
                  all notes to close() apply.
                - similar to 'rm' as defined by POSIX.


    - close
      Purpose:  closes the object
      Format:   close              (in  float timeout = 0.0);
      Inputs:   timeout             seconds to wait
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - the entry instance is closed.
+     Perms:    -
      Throws:   NotImplemented
                IncorrectState
                NoSuccess
```

```
-     Notes:    - 'IncorrectState' is thrown if the object was
-                 closed or removed before.
      Notes:    - any subsequent method call on the object
!                 MUST raise an 'IncorrectState' exception
!                 (apart from DESTRUCTOR and close()).
                - close() can be called multiple times, with no
                  side effects.
+               - if close() is implicitely called in the
+                 DESTRUCTOR, it will never throw an exception.
-               - it is assumed that a session which opened the
-                 instance can also close it - otherwise the
-                 backend entity must have changed its state,
-                 which causes an 'IncorrectState' exception.
                - for resource deallocation semantics, see
!                 Section 2.
!               - for timeout semantics, see Section 2.
+
+
+   // overload permissions because of namespace specific flags
+
+   - permissions_allow
+     Purpose:  enable a permission
+     Format:   permissions_allow    (in  string  id,
+                                     in  int     perm,
+                                     in  int     flags = None);
+     Inputs:   id:                  id to set permission for
+               perm:                permission to enable
+               flags:               mode of operation
+     InOuts:   -
+     Outputs:  -
+     PreCond:  -
+     PostCond: - the permissions are enabled.
+     Perms:    Owner
+     Throws:   NotImplemented
+               BadParameter
+               IncorrectState
+               PermissionDenied
+               AuthorizationFailed
+               AuthenticationFailed
+               Timeout
+               NoSuccess
+     Notes:    - all notes to permissions_allow from the
+                 saga::permissions interface apply.
+               - allowed flags are: 'Recursive', 'Dereference'.
+                 All other flags cause a 'BadParameter'
+                 exception.
```

```
+               - specifying 'Recursive' for a non-directory
+                 causes a 'BadParameter' exception.
+
+
+   - permissions_deny
+     Purpose:  disable a permission flag
+     Format:   permissions_deny     (in  string     id,
+                                     in  int        perm,
+                                     in  int        flags);
+     Inputs:   id:                  id to set permission for
+               perm:                permission to disable
+               flags:               mode of operation
+     InOuts:   -
+     Outputs:  -
+     PreCond:  -
+     PostCond: - the permissions are disabled.
+     Perms:    Owner
+     Throws:   NotImplemented
+               BadParameter
+               IncorrectState
+               PermissionDenied
+               AuthorizationFailed
+               AuthenticationFailed
+               Timeout
+               NoSuccess
+     Notes:    - all notes to permissions_deny from the
+                 saga::permissions interface apply.
+               - allowed flags are: 'Recursive', 'Dereference'.
+                 All other flags cause a 'BadParameter'
+                 exception.
+               - specifying 'Recursive' for a non-directory
+                 causes a 'BadParameter' exception.
```

Class `ns_directory`

`ns_directory` inherits all navigation and manipulation methods from `ns_entry`, but adds some more methods to these sets: instead of `dir.copy (target)` they allow, for example, to do `dir.copy (source, target)`. Other methods added allow to change the cwd of the instance (which changes the values returned by the `get_name()`, `get_cwd()` and `get_url()` inspection methods), and others allow to open new `ns_entry` and `ns_directory` instances (`open()` and `open_dir()`).

For all methods which have the same name as in the `ns_entry` class, the descriptions and semantics defined in `ns_entry` apply, unless noted here otherwise.

```
-   Constructor / Destructor:
-   -------------------------
-
    - CONSTRUCTOR
      Purpose:  create the object
!     Format:   CONSTRUCTOR          (in  session   s,
                                      in  saga::url name,
                                      in  int       flags = None,
                                      out ns_directory obj)
      Inputs:   name:                 initial working dir
                flags:                open mode
!               s:                    session handle for
                                      object creation
+     InOuts:   -
      Outputs:  obj:                  the newly created object
+     PreCond:  -
+     PostCond: - the directory is opened.
+               - 'Owner' of target is the id of the context
+                 use to perform the opereration, if the
+                 directory gets created.
+     Perms:    Exec  for parent directory.
+               Write for parent directory if Create is set.
+               Write for name if Write is set.
+               Read  for name if Read  is set.
      Throws:   NotImplemented
+               IncorrectURL
                BadParameter
                DoesNotExist
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - the semantics of the inherited constructors
                  apply
                - the constructor performs an open of the
                  entry - all notes to the respective open
                  call apply.
!               - the default flags are 'None' (0).
```

```
    - DESTRUCTOR
      Purpose:  destroy the object
      Format:   DESTRUCTOR           (in  ns_directory obj)
      Inputs:   obj:                  the object to destroy
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - the directory is closed.
+     Perms:    -
      Throws:   -
      Notes:    - the semantics of the inherited destructors
                  apply


    Methods for navigation in the name space hierarchy:
    ---------------------------------------------------

    - change_dir
      Purpose:  change the working directory
      Format:   change_dir         (in  saga::url  dir);
      Inputs:   dir:                directory to change to
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - dir is the directory the instance represents.
+     Perms:    Exec for dir.
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
!     Notes:    - if 'dir' can be parsed as URL, but contains an
!                 invalid directory name, a 'BadParameter'
                  exception is thrown.
                - if 'dir' does not exist, a 'DoesNotExist'
                  exception is thrown.
!               - similar to the 'cd' command in the POSIX
!                 shell.


    - list
```

```
      Purpose:  list entries in this directory
!     Format:   list               (in  string name_pattern = ".",
                                    in  int    flags        = None
                                    out array<saga::url> names);
      Inputs:   flags:             flags defining the operation
                                   modus
                name_pattern:      name or pattern to list
+     InOuts:   -
      Outputs:  names:             array of names matching the
                                   name_pattern
+     PreCond:  -
+     PostCond: -
+     Perms:    Query for entries specified by name_pattern.
+               Exec  for parent directories of these entries.
+               Query for parent directories of these entries.
+               Read  for directories specified by name_pattern.
+               Exec  for directories specified by name_pattern.
+               Exec  for parent directories of these directories.
+               Query for parent directories of these directories.
      Throws:   NotImplemented
+               IncorrectURL
                BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
!     Notes:    - if name_pattern is not given (i.e. is an empty
                  string), all entries in the current working
                  directory are listed.
+               - if name_pattern is given and points to a
                  directory, the contents of that directory
+                 are listed.
+               - the name_pattern follows the standard POSIX
+                 shell wildcard specification, as described
+                 above.
                - list does not follow symbolically linked
                  directories, unless the 'Dereference' flag
                  is specified - otherwise list lists symbolic
!                 link entries with a matching name.
                - if the 'DeReference' flag is set, list
                  returns the name of link targets, not of the
                  link entry itself.
!               - the default flags are 'None' (0).
                - other flags are not allowed, and cause a
```

```
                ’BadParameter’ exception.
              - if the name_pattern cannot be parsed, a
!               ’BadParameter’ exception with a descriptive
                error message is thrown.
              - if the name_pattern does not match any entry,
                an empty list is returned, but no exception is
                raised.
              - similar to ’ls’ as defined by POSIX.


    - find
      Purpose:  find entries in the current directory and below
      Format:   find               (in  string  name_pattern,
                                    in  int     flags = Recursive,
                                    out array<saga::url> names);
      Inputs:   name_pattern:       pattern for names of
                                    entries to be found
                flags:              flags defining the operation
                                    modus
+     InOuts:   -
      Outputs:  names:              array of names matching the
                                    name_pattern
+     PreCond:  -
+     PostCond: -
+     Perms:    Read  for cwd.
                Query for entries specified by name_pattern.
+               Exec  for parent directories of these entries.
+               Query for parent directories of these entries.
+               Read  for directories specified by name_pattern.
+               Exec  for directories specified by name_pattern.
+               Exec  for parent directories of these directories.
+               Query for parent directories of these directories.
      Throws:   NotImplemented
                BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
!     Notes:    - find operates recursively below the current
                  working directory if the ’Recursive’ flag is
                  specified (default)
                - find does not follow symbolically linked
                  directories, unless the ’Dereference’ flag
                  is specified - otherwise find lists symbolic
```

```
!                 link entries with a matching name.
-               - if the 'DeReference' flag is set, list
-                 returns the name of link targets, not of the
-                 link entry itself.
!               - the default flags are 'Recursive' (1).
                - other flags are not allowed, and cause a
                  'BadParameter' exception.
!               - the name_pattern follows the standard POSIX
                  shell wildcard specification, as described
                  above.
!               - the matching entries returned are path names
!                 relative to cwd.
                - similar to 'find' as defined by POSIX, but
                  limited to the -name option.


    - exists
      Purpose:  returns true if entry exists, false otherwise
      Format:   exists             (in  saga::url name,
                                    out boolean  exists);
      Inputs:   name:               name to be tested for
                                    existence
+     InOuts:   -
      Outputs:  exists:             boolean indicating existence
                                    of name
+     PreCond:  -
+     PostCond: -
+     Perms:    Query for name.
+               Exec  for name's parent directory.
+               Read  for name's parent directory.
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
!     Notes:    - if 'name' can be parsed as URL, but contains
!                 an invalid entry name, an 'BadParameter'
                  exception is thrown.
                - note that no exception is thrown if the entry
!                 does not exist - the method just returns
                  'false' in this case.
                - similar to 'test -e' as defined by POSIX.
```

```
    - is_dir
!     Purpose:  tests name for being a directory
      Format:   is_dir             (in  saga::url name,
-                                   in  int     flags = None,
                                    out boolean test);
      Inputs:   name:              name to be tested
-               flags:             flags for operation
+     InOuts:   -
      Outputs:  test:              boolean indicating if name
                                   is a directory
+     PreCond:  -
+     PostCond: -
+     Perms:    Query for name.
+               Exec  for name's parent directory.
+               Read  for name's parent directory.
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - returns true if the instance represents
                  a directory entry, false otherwise
                - all notes to the ns_ntry::is_dir() method apply.
-               - the default flags are 'None' (0).
!               - if 'name' can be parsed as URL, but contains
!                 an invalid entry name, an 'BadParameter'
                  exception is thrown.
                - if 'name' is a valid entry name but the entry
!                 does not exist, a 'DoesNotExist' exception is
                  thrown.
-               - if the entry identified by 'name' is a symbolic
-                 link and the 'Dereference' flag is given, the
-                 link target is tested.  If the flag is not
-                 given, the method returns 'false'.
+               - similar to 'test -d' as defined by POSIX.
-               - similar to 'test ! -d' as defined by POSIX.


    - is_entry
```

```
!     Purpose:  tests name for being an ns_entry
      Format:   is_entry           (in  saga::url name,
-                                   in  int       flags = None,
                                    out boolean   test);
      Inputs:   name:              name to be tested
-               flags:             flags for operation
+     InOuts:   -
      Outputs:  test:              boolean indicating if name
                                   is a non-directory entry
+     PreCond:  -
+     PostCond: -
+     Perms:    Query for name.
+               Exec  for name's parent directory.
+               Read  for name's parent directory.
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - all notes to the ns_ntry::is_entry() method
                  apply.
-               - the default flags are 'None' (0).
!               - if 'name' can be parsed as URL, but contains
!                 an invalid entry name, a 'BadParameter'
                  exception is thrown.
                - if 'name' is a valid entry name but the entry
!                 does not exist, a 'DoesNotExist' exception is
                  thrown.
-               - if the entry identified by 'name' is a symbolic
-                 link and the 'Dereference' flag is given, the
-                 link target is tested.  If the flag is not
-                 given, the method returns 'false'.
!               - similar to 'test -f' as defined by POSIX.


    - is_link
!     Purpose:  tests name for being a symbolic link
      Format:   is_link            (in  saga::url name,
-                                   in  int       flags = None,
                                    out boolean   test);
      Inputs:   name:              name to be tested
```

```
-               flags:              flags for operation
+     InOuts:   -
      Outputs:  test:               boolean indicating if name
                                    is a link
+     PreCond:  -
+     PostCond: -
+     Perms:    Query for name.
+               Exec  for name's parent directory.
+               Read  for name's parent directory.
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
                IncorrectState
                DoesNotExist
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - all notes to the ns_ntry::is_link() method
                  apply.
-               - the default flags are 'None' (0).
!               - if 'name' can be parsed as URL, but contains
!                 an invalid entry name, a 'BadParameter'
                  exception is thrown.
                - if 'name' is a valid entry name but the entry
!                 does not exist, a 'DoesNotExist' exception is
                  thrown.
-               - if the entry identified by 'name' is a symbolic
-                 link and the 'Dereference' flag is given, the
-                 link target is tested.  If the flag is not
-                 given, the method returns 'false'.
                - similar to 'test -L' as defined by POSIX.


    - read_link
      Purpose:  returns the name of the link target
      Format:   read_link          (in  saga::url name,
                                    out saga::url link);
      Inputs:   name:               name to be resolved
+     InOuts:   -
      Outputs:  link:               resolved name
+     PreCond:  -
+     PostCond: -
+     Perms:    Query for name.
+               Exec  for name's parent directory.
```

```
+               Read  for name's parent directory.
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - all notes to ns_entry::read_link() apply
!               - if 'name' can be parsed as URL, but contains
!                 an invalid entry name, a 'BadParameter'
                  exception is thrown.
                - if 'name' does not exist, a 'DoesNotExist'
                  exception is thrown.



    Iterate over large directories:
    -------------------------------

    - get_num_entries
      Purpose:  gives the number of entries in the directory
      Format:   get_num_entries    (out int   num);
      Inputs:   -
+     InOuts:   -
      Outputs:  num:                number of entries in the
                                    directory
+     PreCond:  -
+     PostCond: -
+     Perms:    Query for cwd.
+               Exec  for cwd.
+               Read  for cwd.
      Throws:   NotImplemented
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - at the time of using the result of this call,
                  the actual number of entries may already have
                  changed (no locking is implied)
                - vaguely similar to 'opendir'/'readdir' (2) as
```

```
                  defined by POSIX.


    - get_entry
      Purpose:  gives the name of an entry in the directory
                based upon the enumeration defined by
                get_num_entries
      Format:   get_entry          (in  int       entry,
                                    out saga::url name);
      Inputs:   entry:              index of entry to get
+     InOuts:   -
      Outputs:  name:               name of entry at index
+     PreCond:  -
+     PostCond: -
+     Perms:    Query for cwd.
+               Exec  for cwd.
+               Read  for cwd.
      Throws:   NotImplemented
                IncorrectState
                DoesNotExist
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - '0' is the first entry
                - there is no sort order implied by the
                  enumeration, however an underlying
                  implementation MAY choose to sort the entries
                - subsequent calls to get_entry and/or
                  get_num_entries may return inconsistent data,
                  i.e. no locking or state tracking is implied.
                  In particular, an index may be invalid - a
                  'DoesNotExist' exception is then thrown (not a
                  'BadParameter' exception).
                - vaguely similar to 'opendir'/'readdir' (2) as
                  defined by POSIX.

-
-   Methods for managing access control lists:
-   ------------------------------------------
-
-   - set_acl
-     Purpose:  set access control list for this entry
-     Format:   set_acl            (in  string   name,
-                                   in  string   dn,
```

```
-                                  in  int      acl,
-                                  in  int      flags = None);
-     Inputs:   name:              entry to set ACLs for
-               dn:                DN to set ACLs for
-               flags:             flags defining the operation
-                                  modus
-     Outputs:  -
-     Perms:    -
-     Throws:   NotImplemented
-               IncorrectURL
-               BadParameter
-               DoesNotExist
-               IncorrectState
-               PermissionDenied
-               AuthorizationFailed
-               AuthenticationFailed
-               Timeout
-               NoSuccess
-     Notes:    - all notes to the ns_entry::set_acl() method
-                 apply.
-               - the default flags are 'None' (0).
-               - if 'name' can be parsed as URL, but contains
-                 an invalid entry name, a 'BadParameter'
-                 exception is thrown.
-               - if 'name' is a valid entry name but the entry
-                 does not exist, a 'DoesNotExist' exception is
-                 thrown.
-
-
-   - get_acl
-     Purpose:  get access control list for this entry
-     Format:   get_acl            (in  string   name,
-                                   in  string   dn,
-                                   in  int      flags = None,
-                                   out int      acl);
-     Inputs:   dn:                entry to get ACLs for
-               dn:                DN to get ACLs for
-               flags:             flags defining the operation
-                                  modus
-     Outputs:  acl:               OR'ed ACLs set on the
-                                  entity, for the specified dn
-     Perms:    -
-     Throws:   NotImplemented
-               IncorrectURL
-               BadParameter
-               DoesNotExist
```

```
-               IncorrectState
-               PermissionDenied
-               AuthorizationFailed
-               AuthenticationFailed
-               Timeout
-               NoSuccess
-     Notes:    - all notes to the ns_entry::get_acl() method
-                 apply.
-               - the default flags are 'None' (0).
-               - if 'name' can be parsed as URL, but contains
-                 an invalid entry name, a 'BadParameter'
-                 exception is thrown.
-               - if 'name' is a valid entry name but the entry
-                 does not exist, a 'DoesNotExist' exception is
-                 thrown.
-
-
-   - list_dn
-     Purpose:  list all DN's for which ACLs are set.
-     Format:   list_dn            (in  string name,
-                                   in  int    flags = None,
-                                   out array<string>  dn);
-     Inputs:   name:               entry to list DNs foreration
-               flags:              flags defining the operation
-     Outputs:  dn:                 list of DNs for which ACLs
-                                   are set on the entry
-     Perms:    -
-     Throws:   NotImplemented
-               IncorrectURL
-               BadParameter
-               DoesNotExist
-               IncorrectState
-               PermissionDenied
-               AuthorizationFailed
-               AuthenticationFailed
-               Timeout
-               NoSuccess
-     Notes:    - all notes to the ns_entry::get_dn() method
-                 apply.
-               - the default flags are 'None' (0).
-               - if 'name' can be parsed as URL, but contains
-                 an invalid entry name, a 'BadParameter'
-                 exception is thrown.
-               - if 'name' is a valid entry name but the entry
-                 does not exist, a 'DoesNotExist' exception is
-                 thrown.
```

```
-
-

    Management of name space entries:
    ---------------------------------

    - copy
      Purpose:  copy the entry to another part of the name space
      Format:   copy               (in  saga::url  source,
                                    in  saga::url  target,
                                    in  int     flags = None);
      Inputs:   source:             name to copy
                target:             name to copy to
                flags:              flags defining the operation
                                    modus
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - an identical copy of source exists at target.
+               - 'Owner' of target is the id of the context
+                 used to perform the opereration if target gets
+                 created.
+     Perms:    Query for source.
+               Exec  for source's parent directory.
+               Query for target.
+               Query for target's parent directory.
+               Exec  for target's parent directory.
+               Write for target
+                     if  target does exist.
+               Write for target's parent directory
+                     if  target does not exist.
      Throws:   NotImplemented
+               IncorrectURL
                BadParameter
                AlreadyExists
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - all notes to the ns_entry::copy() method
                  apply.
!               - the default flags are 'None' (0).
!               - if 'name' can be parsed as URL, but contains
```

```
!                 an invalid entry name, a 'BadParameter'
                  exception is thrown.
                - if 'name' is a valid entry name but the entry
!                 does not exist, a 'DoesNotExist' exception is
                  thrown.


    - link
!     Purpose:  create a symbolic link from the target entry to
!               the source entry so that any reference to the
                target refers to the source entry
      Format:   link               (in  saga::url  source,
                                    in  saga::url  target,
                                    in  int     flags = None);
      Inputs:   source:             name to link
                target:             name to link to
                flags:              flags defining the operation
                                    modus
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - a symbolic link to source exists at target.
+               - 'Owner' of target is the id of the context
+                 used to perform the opereration if target gets
+                 created.
+     Perms:    Query for source.
+               Exec  for source's parent directory.
+               Query for target.
+               Query for target's parent directory.
+               Exec  for target's parent directory.
+               Write for target
+                     if  target does exist.
+               Write for target's parent directory
+                     if  target does not exist.
      Throws:   NotImplemented
+               IncorrectURL
                BadParameter
                AlreadyExists
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - all notes to the ns_entry::link() method
```

```
                apply.
+               - if the 'Recursive' flag is defined, the source
+                 is recursively linked if it is a directory;
+                 otherwise this flag is ignored.
+               - if the 'Dereference' flag is specified, the
+                 method applies to the link target of source.
+                 The flag causes a 'BadParameter' exception if
+                 source is not a link.
+               - if the the target already exists, the
+                 'Overwrite' flag must be specified, otherwise
+                 an 'AlreadyExists' exception is thrown.
!               - the default flags are 'None' (0).
+               - other flags are not allowed on this method,
+                 and cause a 'BadParameter' exception.
!               - if 'source' can be parsed as URL, but contains
!                 an invalid entry name, a 'BadParameter'
                  exception is thrown.
                - if 'source' is a valid entry name but the entry
!                 does not exist, a 'DoesNotExist' exception is
                  thrown.


    - move
      Purpose:  rename source to target, or move source to
!               target if target is a directory.
      Format:   move               (in  saga::url  source,
                                    in  saga::url  target,
                                    in  int     flags = None);
      Inputs:   source:             name to move
                target:             name to move to
                flags:              flags defining the operation
                                    modus
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - an identical copy of source exists at target.
+               - source is removed.
+               - 'Owner' of target is the id of the context
+                 used to perform the opereration if target gets
+                 created.
+     Perms:    Query for source.
+               Write for source.
+               Exec  for source's parent directory.
+               Write for source's parent directory.
+               Query for target.
+               Exec  for target's parent directory.
```

```
+               Write for target
+                     if  target does exist.
+               Write for target's parent directory
+                     if  target does not exist.
      Throws:   NotImplemented
+               IncorrectURL
                BadParameter
                AlreadyExists
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - all notes to the ns_entry::move() method
                  apply.
+               - if the 'Recursive' flag is defined, the source
+                 is recursively copied if it is a directory;
+                 otherwise this flag is ignored.
+               - if the 'Dereference' flag is specified, the
+                 method applies to the link target of source.
+                 The flag causes a 'BadParameter' exception if
+                 source is not a link.
+               - if the the target already exists, the
+                 'Overwrite' flag must be specified, otherwise
+                 an 'AlreadyExists' exception is thrown.
!               - the default flags are 'None' (0).
+               - other flags are not allowed on this method,
+                 and cause a 'BadParameter' exception.
!               - if 'source' can be parsed as URL, but contains
!                 an invalid entry name, a 'BadParameter'
                  exception is thrown.
                - if 'source' is a valid entry name but the entry
!                 does not exist, a 'DoesNotExist' exception is
                  thrown.
                - moving any parent or the current directoy
                  (e.g.  '.', '..' etc.) is not allowed, and
                  throws a 'BadParameter' exception


    - remove
      Purpose:  removes the entry
      Format:   remove             (in  saga::url  target,
                                    in  int     flags = None);
      Inputs:   target:             entry to be removed
```

```
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - source is removed.
+               - source is closed if it refers to the cwd.
+     Perms:    Query for source.
+               Write for source.
+               Exec  for source's parent directory.
+               Write for source's parent directory.
      Throws:   NotImplemented
+               IncorrectURL
                BadParameter
                AlreadyExists
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - all notes to the ns_entry::remove() method
                  apply.
+               - if the 'Recursive' flag is defined, the source
+                 is recursively removed if it is a directory;
+                 otherwise this flag is ignored.
+               - if the 'Dereference' flag is specified, the
+                 method applies to the link target of source.
+                 The flag causes a 'BadParameter' exception if
+                 source is not a link.
!               - the default flags are 'None' (0).
+               - other flags are not allowed on this method,
+                 and cause a 'BadParameter' exception.
!               - if 'source' can be parsed as URL, but contains
!                 an invalid entry name, a 'BadParameter'
                  exception is thrown.
                - if 'source' is a valid entry name but the entry
!                 does not exist, a 'DoesNotExist' exception is
                  thrown.
                - removing any parent or the current directoy
                  (e.g.  '.', '..' etc.) is not allowed, and
                  throws a 'BadParameter' exception


    - make_dir
      Purpose:  creates a new directory
      Format:   make_dir           (in  saga::url  target,
```

```
                                    in  int     flags = None);
      Inputs:   target:             directory to create
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - 'Owner' of target is the id of the context
+                 used to perform the opereration if target gets
+                 created.
+     Perms:    Exec  for target's parent directory.
+               Write for target's parent directory.
+               Write for target if Write is set.
+               Read  for target if Read  is set.
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
                AlreadyExists
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - if the parent directory or directories do not
!                 exist, the 'CreateParents' flag must be set
!                 or a 'DoesNotExist' exception is thrown.
!                 If set, the parent directories are created as
                  well.
                - an 'AlreadyExists' exception is thrown if the
                  directory already exists and the 'Exclusive'
                  flag is given.
!               - the default flags are 'None' (0).
                - other flags are not allowed on this method,
!                 and cause a 'BadParameter' exception.
!               - if 'target' can be parsed as URL, but contains
!                 an invalid entry name, a 'BadParameter'
                  exception is thrown.
                - similar to 'mkdir' (2) as defined by POSIX.


    - open_dir
      Purpose:  creates a new ns_directory instance
      Format:   open_dir           (in  saga::url name,
                                    in  int       flags = None,
                                    out ns_directory dir);
      Inputs:   name:               directory to open
```

```
                flags:              flags defining the operation
                                    modus
+     InOuts:   -
      Outputs:  dir:                opened directory instance
+     PreCond:  -
+     PostCond: - the session of the returned instance is that of
+               the calling instance.
+               - 'Owner' of name is the id of the context
+               used to perform the opereration if name gets
+               created.
+               - the namespace directory is created if it
+               does not yet exist, and the Create is set.
+     Perms:    Exec  for name's parent directory.
+               Write for name's parent directory if Create is set.
+               Write for name if Write is set.
+               Read  for name if Read  is set.
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
                AlreadyExists
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - the cwd of the new dir object instance is set
                  to 'name'
                - a 'DoesNotExist' exception is thrown if 'name'
!                 does not exist and the 'Create' flag is not
                  given.
                - a 'AlreadyExist' exception is thrown if 'name'
!                 does exist and the 'Create' flag and the
                 'Exclusive' flag are given.
                - no exception is thrown if 'name' does exist and
                  the 'Create' flag is given, and the 'Exclusive'
                  flag is not given.
!               - if the 'Create' flag is given, all notes to the
                  ns_directory::make_dir() method apply.
!               - the default flags are 'None' (0).
                - other flags are not allowed on this method,
!                 and cause a 'BadParameter' exception.
                - 'name' is always deeply dereferenced, however,
                  the cwd is still set to 'name', and not to the
                  value of the link target.
```

```
+               - parent directories are created on the fly if
+                 the 'CreateParents' and 'Create' flag are both
+                 given, if they don't exist.
!               - if 'name' can be parsed as URL, but contains
!                 an invalid directory name, a 'BadParameter'
                  exception is thrown.
-               - in the asynchronous case, the resulting
-                 directory instance is passed as reference.  If
-                 that directory instance belongs to the default
-                 SAGA session, its session will be changed to
-                 the current session.  If that directory
-                 instance session is not the default session,
-                 and is different from the current session,
-                 an 'IncorrectSession' exception is thrown.


    - open
      Purpose:  creates a new ns_entry instance
      Format:   open               (in  saga::url name,
                                    in  int       flags = None,
                                    out ns_entry  entry);
      Inputs:   name:               entry
                flags:              flags defining the operation
                                    modus
+     InOuts:   -
      Outputs:  entry:              opened entry instance
+     PreCond:  -
+     PostCond: - the session of the returned instance is that
+                 of the calling instance.
+               - 'Owner' of name is the id of the context
+                 used to perform the opereration if name gets
+                 created.
+               - the namespace entry is created if it does not
+                 yet exist, and the CREATE flag is specified.
+     Perms:    Exec  for name's parent directory.
+               Write for name's parent directory if Create is set.
+               Write for name if Write is set.
+               Read  for name if Read  is set.
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
                AlreadyExists
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
```

```
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - a 'BadParameter' exception is thrown if 'name'
                  points to a directory, or is an invalid entry
                  name.
                - a 'DoesNotExist' exception is thrown if 'name'
!                 does not exist, and the 'Create' flag is not
                  given.
                - a 'AlreadyExists' exception is thrown if 'name'
!                 does exist, and the 'Create' and 'Exclusive'
                  flags are given.
                - 'name' is always deeply dereferenced, the cwd,
                  however, is not changed to the link targets
                  cwd.
+               - parent directories are created on the fly if
+                 the 'CreateParents' and 'Create' flag are both
+                 given, if they don't exist.
                - the entry is locked on open if the 'Lock' flag
                  is given.  If the entry is already in a locked
                  state, the open will fail and a descriptive
                  error will be issued.  If a entry is opened in
                  locked mode, any other open on that entry MUST
                  fail with a 'NoSuccess' exception if the 'Lock'
                  flag is given.  Note that a entry can be opened
!                 in unlocked mode, and then in locked mode,
!                 without an error getting raised.  The
                  application programmer must take precautions
!                 to avoid such situations.  The lock will get
                  removed on destruction of the entry object, and
                  also on close.  If an implementation does not
!                 support locking, a descriptive 'BadParameter'
!                 exception MUST get thrown if the 'Lock' flag
!                 is given.  Read-locks and Write-locks are not
+                 distinguished.
!               - the default flags are 'None' (0).
                - other flags are not allowed on this method,
!                 and cause a 'BadParameter' exception.
-               - in the asynchronous case, the resulting
-                 entry instance is passed by reference.  If
-                 that entry instance belongs to the default
-                 SAGA session, its session will be changed to
-                 the current session.  If that entry
-                 instance session is not the default session,
-                 and is different from the current session,
-                 an 'IncorrectSession' exception is thrown.
```

```
                - similar to 'open' (2) as defined by POSIX.
+
+
+   Management of name space entries - wildcard versions:
+   -----------------------------------------------------
+
+   - copy
+     Purpose:  copy the entry to another part of the name space
+     Format:   copy               (in  string     source,
+                                   in  saga::url  target,
+                                   in  int        flags = None);
+     Notes:    - the syntax and semantics of this call is
+                 identical to its URL based couterpart.
+               - the 'source' string can contain wildcards, as
+                 described above.
+               - on error conditions on any of the expanded
+                 list of source entries, the respective error
+                 described in the URL version of the method is
+                 thrown - the state of the operations on the
+                 other elements of the expanded entry list is
+                 undefined.
+               - if source expands to multiple entries, then the
+                 target URL MUST specify a directory -
+                 otherwise a 'BadParameter' exception is thrown.
+
+
+   - link
+     Purpose:  create a symbolic link from the target entry to
+               the source entry so that any reference to the
+               target refers to the source entry
+     Format:   link               (in  string     source,
+                                   in  saga::url  target,
+                                   in  int        flags = None);
+     Notes:    - the syntax and semantics of this call is
+                 identical to its URL based couterpart.
+               - the 'source' string can contain wildcards, as
+                 described above.
+               - on error conditions on any of the expanded
+                 list of source entries, the respective error
+                 described in the URL version of the method is
+                 thrown - the state of the operations on the
+                 other elements of the expanded entry list is
+                 undefined.
+               - if source expands to multiple entries, then the
+                 target URL MUST specify a directory -
+                 otherwise a 'BadParameter' exception is thrown.
```

```
+
+   - move
+     Purpose:  moves sources to a target directory.
+     Format:   move               (in  string     source,
+                                   in  saga::url  target,
+                                   in  int        flags = None);
+     Notes:    - the syntax and semantics of this call is
+                 identical to its URL based couterpart.
+               - the 'source' string can contain wildcards, as
+                 described above.
+               - on error conditions on any of the expanded
+                 list of source entries, the respective error
+                 described in the URL version of the method is
+                 thrown - the state of the operations on the
+                 other elements of the expanded entry list is
+                 undefined.
+               - if source expands to multiple entries, then the
+                 target URL MUST specify a directory -
+                 otherwise a 'BadParameter' exception is thrown.
+
+   - remove
+     Purpose:  removes entries
+     Format:   remove             (in  string  target,
+                                   in  int     flags = None);
+     Notes:    - the syntax and semantics of this call is
+                 identical to its URL based couterpart.
+               - the 'source' string can contain wildcards, as
+                 described above.
+               - on error conditions on any of the expanded
+                 list of source entries, the respective error
+                 described in the URL version of the method is
+                 thrown - the state of the operations on the
+                 other elements of the expanded entry list is
+                 undefined.
+
+
+   // overload permissions because of namespace specific flags
+
+   - permissions_allow
+     Purpose:  enable a permission
+     Format:   permissions_allow    (in  saga::url target,
+                                     in  string    id,
+                                     in  int       perm,
+                                     in  int       flags = None);
+     Inputs:   target:               entry to set permissions for
+               id:                   id to set permission for
```

```
+               perm:                 permission to enable
+               flags:                mode of operation
+     InOuts:   -
+     Outputs:  -
+     PreCond:  -
+     PostCond: - the permissions are enabled.
+     Perms:    Owner of target
+     Throws:   NotImplemented
+               IncorrectURL
+               BadParameter
+               IncorrectState
+               PermissionDenied
+               AuthorizationFailed
+               AuthenticationFailed
+               Timeout
+               NoSuccess
+     Notes:    - all notes to permissions_allow from the
+                 saga::permissions interface apply.
+               - allowed flags are: 'Recursive', 'Dereference'.
+                 All other flags cause a 'BadParameter'
+                 exception.
+               - specifying 'Recursive' for a non-directory
+                 causes a 'BadParameter' exception.
+
+
+   - permissions_deny
+     Purpose:  disable a permission flag
+     Format:   permissions_deny     (in  saga::url target,
+                                     in  string    id,
+                                     in  int       perm,
+                                     in  int       flags = None);
+     Inputs:   target:               entry to set permissions for
+               id:                   id to set permission for
+               perm:                 permission to disable
+               flags:                mode of operation
+     InOuts:   -
+     Outputs:  -
+     PreCond:  -
+     PostCond: - the permissions are disabled.
+     Perms:    Owner of target
+     Throws:   NotImplemented
+               IncorrectURL
+               BadParameter
+               IncorrectState
+               PermissionDenied
+               AuthorizationFailed
```

```
+               AuthenticationFailed
+               Timeout
+               NoSuccess
+     Notes:    - all notes to permissions_deny from the
+                 saga::permissions interface apply.
+               - allowed flags are: 'Recursive', 'Dereference'.
+                 All other flags cause a 'BadParameter'
+                 exception.
+               - specifying 'Recursive' for a non-directory
+                 causes a 'BadParameter' exception.
+
+
+   // permissions calls - wildcard versions
+
+   - permissions_allow
+     Purpose:  enable a permission
+     Format:   permissions_allow    (in  string  target,
+                                     in  string  id,
+                                     in  int     perm,
+                                     in  int     flags = None);
+     Notes:    - the syntax and semantics of this call is
+                 identical to its URL based couterpart.
+               - the 'source' string can contain wildcards, as
+                 described above.
+               - on error conditions on any of the expanded
+                 list of source entries, the respective error
+                 described in the URL version of the method is
+                 thrown - the state of the operations on the
+                 other elements of the expanded entry list is
+                 undefined.
+
+
+   - permissions_deny
+     Purpose:  disable a permission flag
+     Format:   permissions_deny     (in  string  target,
+                                     in  string  id,
+                                     in  int     perm,
+                                     in  int     flags = None);
+     Notes:    - the syntax and semantics of this call is
+                 identical to its URL based couterpart.
+               - the 'source' string can contain wildcards, as
+                 described above.
+               - on error conditions on any of the expanded
+                 list of source entries, the respective error
+                 described in the URL version of the method is
+                 thrown - the state of the operations on the
```

```
+                 other elements of the expanded entry list is
+                 undefined.
```

4.2.4 Examples:

Code Example

```
More examples are given in the File and Logical_File packages.

Example: provide recursive directory listing for a given
         directory

Note:    - check for '.' and '..' recursion are left as an
           exercise to the reader.
         - string operations and printf statements are
           obviously simplified.

+-------------------------------------------------------------+
 // c++ example
 std::string indent (int indent)
 {
   std::string s = "  ";

   for (int i = 0; i < indent; i++, s += "  ");

   return (s);
 }

 void list_dir (saga::url url,
                int       indent = 0)
 {
   try
   {
     // create directory and iterate over entries
     saga::ns_dir dir (url);

     printf ("\n%s ---> %s\n", indent (indent), url.get_url ());

     for ( int i = 0; i < dir.get_num_entries (); i++ )
     {
       char   type = '?';
       string info = "";

       // get name of next entry
       saga::url name = dir.get_entry (i);

       // get type and other infos
       if ( dir.is_link (name) )
```

```
42          {
43            // check where link points to
44            if (dir.exists(dir.read_link (name))){info=" ---> ";}
45            else                                 {info=" -|-> ";}
46            info += dir.read_link (name);
47            type  = 'l';
48          }
49          else if (dir.is_entry(name)){ type = 'f';            }
50          else if (dir.is_dir  (name)){ type = 'd'; info = "/";}
51
52          printf ("%s > %3d - %s - %s%s\n",
53                  indent (indent), i + 1,
54                  type, name.get_cstr (), info);
55
56          // recursion on directories
57          if ( dir.is_dir (name) )
58          {
59            list_dir (name, indent++);
60          }
61        }
62
63        printf ("\n%s <--- %s\n", indent (indent), url.get_url ());
64      }
65
66      // catch all errors - see elsewhere for better examples
67      // of error handling in SAGA
68      catch ( const saga::exception & e )
69      {
70        std::cerr << "Oops! SAGA exception: "
71                  << e.get_message ()
72                  << std::endl;
73      }
74
75      return;
76    }
77
78  +-------------------------------------------------------------+
79
80    // a C++ example for ACL management
81
82    ------------------------------------------------
83    | REMOVED - see permissions package for new example |
84    ------------------------------------------------
85
86    {
87      // allow short forms of flags
88      using namespace saga::ns_entry;
89
90      std::string dn_user  = "O=dutchgrid, O=vu, CN=Joe Doe";
91      std::string dn_group = "O=dutchgrid, O=vu, CN=*";
```

```
92
93      // open file (default: Read only)
94      saga::file f (url);
95
96      // set ACL restrictions for file.  The ACL set is
97      // performed with the permissions of the session context
98      f.set_acl (dn_user,  ACL_Read | ACL_Write);
99      f.set_acl (dn_group, ACL_Read);
100
101     // check if acls allow writes with our current session
102     // contexts
103     if ( f.get_acl () & ACL_Write )
104     {
105       saga::file f_2 (url, ReadWrite);
106
107       f_2.write ("data", 4);
108     }
109   }
```

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.

The described classes are **syntactically** and semantically POSIX oriented [21, 22, 23]. *Executing* large numbers of simple POSIX-like remote data access operations *is*, however, prone to latency related performance problems. To allow for efficient implementations, the presented API borrows ideas from GridFTP and other specifications which are widely used for remote data access. These **extensions** should be seen as just that: optimizations. Implementations of this package MUST implement the POSIX-like `read()`, `write()` and `seek()` methods, and MAY implement the additional optimized methods (a 'NotImplemented' *exception* MUST be thrown if these are not implemented). The optimizations included here are:

Scattered I/O Scattered I/O operations are already defined by POSIX, as `readv()` and `writev()`. Essentially, these methods represent **v**ector versions of the standard POSIX `read()/write()` methods; the **arguments** are, *basically*, vectors of instructions *to execute*, and buffers to operate **upon**. In other words, `readv()` and `writev()` can be regarded as specialized bulk methods, which cluster multiple I/O operations into a single operation. Advantage of such an approach are that it is easy to implement, is very close to the original POSIX I/O in semantics, and in some cases even very fast. Disadvantages are that for many small I/O operations (a common occurence in SAGA use cases), the description of the I/O operations can be larger than the sent, returned or received data.

Pattern-Based I/O (FALLS) One approach to address the bandwith limitation of scattered I/O is to describe the required I/O operations at a more abstract level. Regularly repeating patterns of binary data can be described by the so-called 'Family of Line Segments' (FALLS) [14]. The pattern-based I/O routines in SAGA use such descriptions to reduce the bandwidth limitation of scattered I/O. The **advantage** of such an approach is that it targets very common data access patterns (at least those very commonly found in SAGA use cases). The disadvantages are that FALLS is a paradigm not widely known or used, and that FALLS is by definition, limited to regular patterns of data,

and hence is inefficient for more randomized data access.

FALLS (FAmiLy of Line Segments) were originally introduced for transformations in parallel computing. There is also a parallel filesystem which uses FALLS to describe the file layout. They can be used to describe regular subsets of arrays with a very compact syntax.

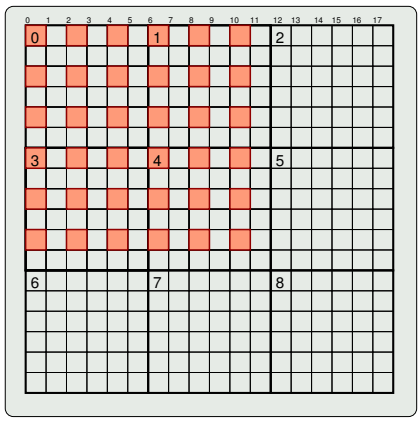

Figure 5: The highlighted elements are defined by `"(0,17,36,6,(0,0,2,6))"`.

FALLS pattern are formed as 5-tuples: `"(from,to,stride,rep,(pat))"`. The `from` element defines the starting offset for the first pattern unit, `to` defines the finishing offset of the first pattern unit, `stride` defines the distance between consecutive pattern units (begin to begin), and `rep` defines the number of repetitions of the pattern units. The optional 5th element `pat` allows to defines nested **patterns**, where the internal pattern defines the unit the outer pattern is applied to (by default that is one byte). As an example: the following FALLS describe the highlighted elements of the matrix in Fig 5: `"(0,17,36,6,(0,0,2,6))"`: the inner pattern describes a pattern unit of one byte length (from 0 to 0), with a distance of 2 to the next application, and 6 repetitions. These are the 6 bytes per line which are marked *red*. The outer pattern defines the repeated application of the inner pattern, starting at 0, ending at 17 (end of line), distance of 36 (to begin of next but one line), and repetition of 6.

Extended I/O GridFTP (which was designed for a similar target domain) introduced an additional remote I/O paradigm, that of Extended I/O operations.

In essence, the Extended I/O paradigm allows the formulation of I/O requests using custom strings, which are not interpreted on the client but on the server side; these can be expanded to **arbitrarily** complex sets of I/O operations. The type of I/O request encoded in the string is called `mode`. A server may support one or many of these extended I/O modes. Whereas the approach is very flexible and powerful and has proven its usability in GridFTP, a disadvantage is that it requires very specific infrastructure to function, i.e. it requires a remote server instance which can interpret opaque client requests. Additionally, no client side checks or optimizations on the I/O requests are possible. Also, the application programmer needs to estimate the size of the data to be returned in advance, which in some cases is very difficult.

The three described operations have, if compared to each other, increasing semantic flexibility, and are increasingly powerful for specific use cases. However, they are also increasingly difficult to implement and support in a generic fashion. It is up to the SAGA implementation and the specific use cases, to determine the level of I/O abstraction that serves the application best and that can be best supported in the target environment.

4.3.1 Specification

```
  package saga.file
  {
    enum flags
    {
!     None           =    0, // same as in namespace::flags
!     Overwrite      =    1, // same as in namespace::flags
!     Recursive      =    2, // same as in namespace::flags
!     Dereference    =    4, // same as in namespace::flags
!     Create         =    8, // same as in namespace::flags
!     Exclusive      =   16, // same as in namespace::flags
!     Lock           =   32, // same as in namespace::flags
!     CreateParents  =   64, // same as in namespace::flags
      Truncate       =  128,
      Append         =  256,
      Read           =  512,
      Write          = 1024,
      ReadWrite      = 1536, // Read | Write
      Binary         = 2048
    }


    enum seek_mode
    {
      Start      =   1,
      Current    =   2,
      End        =   3
    }


-   struct ivec
-   {
-     int           offset;    // position of data  to r/w
-     int           leng_in;   // number   of bytes to r/w
-     array<byte>   buffer;    // data              to r/w
```

```
-     int           leng_out;  // number   of bytes    r/w
-   }
-
+   class iovec : extends saga::buffer
+              // from buffer saga::object
+              // from object saga::error_handler
+   {
+     CONSTRUCTOR (in array<byte>          data   = "",
+                  in    int               size   = 0,
+                  in    int               offset = 0,
+                  in    int               len_in = size,
+                  out   buffer            obj);
+
+     set_offset  (in    int               offset);
+     get_offset  (out   int               offset);
+
+     set_len_in  (in    int               len_in);
+     get_len_in  (out   int               len_in);
+
+     get_len_out (out   int               len_out);
+   }

    class file : extends        saga::ns_entry,
-                implements     saga::attributes
              // from ns_entry  saga::object
              // from ns_entry  saga::async
+             // from ns_entry  saga::permissions
              // from object    saga::error_handler
    {
!     CONSTRUCTOR (in    session           s,
                   in    saga::url         name,
                   in    int               flags = Read,
!                  out   file              obj      );
!     DESTRUCTOR  (in    file              obj      );

      // inspection
      get_size    (out   int               size     );

      // POSIX-like I/O
-     read        (inout array<byte>       buffer,
+     read        (inout buffer            buf,
!                  in    int               len_in = -1,
                   out   int               len_out  );
-     write       (in    array<byte>       buffer,
+     write       (in    buffer            buf,
!                  in    int               len_in = -1,
```

```
                 out   int               len_out  );
      seek       (in   int               offset,
                  in   seek_mode         whence,
                  out  int               position );

      // scattered I/O
!     read_v     (inout array<iovec>     iovecs  );
!     write_v    (inout array<iovec>     iovecs  );

      // pattern-based I/O
      size_p     (in   string            pattern,
                  out  int               size     );
      read_p     (in   string            pattern,
-                 inout array<byte>       buffer,
+                 inout buffer            buf,
                  out  int               len_out  );
      write_p    (in   string            pattern,
-                  in   array<byte>       buffer,
+                  in   buffer            buf,
                  out  int               len_out  );

      // extended I/O
      modes_e    (out  array<string>     emodes   );
+     size_e     (in   string            emode,
+                 in   string            spec,
+                 out  int               size     );
      read_e     (in   string            emode,
                  in   string            spec,
-                 inout array<byte>       buffer,
+                 inout buffer            buf,
                  out  int               len_out  );
      write_e    (in   string            emode,
                  in   string            spec,
-                  in   array<byte>       buffer,
+                  in   buffer            buf,
                  out  int               len_out  );
-
-     // Attributes:
-     //   name:  Blocking
-     //   desc:  defines if file I/O is blocking or
-     //          non-blocking
-     //   mode:  ReadWrite
-     //   type:  Bool
-     //   value: True
-     //   note:  optional; I/O must be blocking if
-     //          attribute is absent
```

```
    }
  }


    class directory : extends            saga::ns_directory
                  // from ns_directory   saga::ns_entry
                  // from ns_entry       saga::object
                  // from ns_entry       saga::async
+                 // from ns_entry       saga::permissions
                  // from object         saga::error_handler
    {
!     CONSTRUCTOR (in    session           s,
                   in    saga::url         name,
                   in    int               flags = Read,
!                  out   directory         obj      );
!     DESTRUCTOR  (in    directory         obj      );

      // inspection methods
      get_size    (in    saga::url         name,
                   in    int               flags = None,
                   out   int               size     );
      is_file     (in    saga::url         name,
                   in    int               flags = None,
                   out   boolean           test     );

      // factory-like methods
      open_dir    (in    saga::url         name,
                   in    int               flags = Read,
                   out   directory         dir      );

      open        (in    saga::url         name,
                   in    int               flags = Read,
                   out   file              file     );
    }
```

4.3.2 Specification Details

Enum `flags`

The ***`flags`*** *enum is inherited from the* ***`namespace`*** *package. A number of file specific flags are added to it. All added flags are used for the opening of* ***`file`*** *and* ***`directory`*** *instances, and are not applicable to the operations inherited from the* ***`namespace`*** *package.*

`Truncate`
: *Upon opening, the file is truncated to length `0`, i.e. a following `read()` operation will never find any data in the file. That flag does not apply to directories.*

`Append`
: *Upon opening, the file pointer is set to the end of the file, i.e. a following `write()` operation will extend the size of the file. That flag does not apply to directories.*

`Read`
: *The file or directory is opened for reading – that does not imply the ability to write to the file or directory.*

`Write`
: *The file or directory is opened for writing – that does not imply the ability to read from the file or directory.*

`ReadWrite`
: *The file or directory is opened for reading and writing.*

`Binary`
: *Some operating systems (notably windows based systems) distinguish between binary and non-binary modes – this flag mimics that behaviour.*

Class `iovec`

The `iovec` class inherits the `saga::buffer` class, and three additional state attributes: `offset`, `len_in` and `len_out` (with the latter one being read-only). With that addition, the new class can be used very much the same way as the `iovec` structure defined by POSIX for `readv/writev`, with the buffer `len_in` beeing interpreted as the POSIX `iov_len`, i.e. the number of bytes to read/write.

If `len_in` is not specified, that length is set to the size of the buffer. It is a `BadParameter` error if `len_in` is specified to be larger than size, for application managed buffers (see Section 3.4 for details on buffer memory management). Before an `iovec` instance is used, it's `len_in` MUST be set to a non-zero value; otherwise it's use will cause a `BadParameter` exception.

After a `read_v()` or `write_v()` operations completes, `len_out` will report the number of bytes read. Before completion, the SAGA implementation MUST report `len_out` to be `-1`.

```
+   - CONSTRUCTOR
+     Purpose:  create an iovec instance
```

```
+     Format:   CONSTRUCTOR     (in  array<byte> data   = "",
+                                in  int         size   = -1,
+                                in  int         offset = 0,
+                                in  int         len_in = size,
+                                out iovec       obj);
+     Inputs:   type:                 data to be used
+               size:                 size of data to be used
+               offset                offset for I/O operation
+               len_in:               number of bytes to read
+                                     or write on read_v/write_v
+     InOuts:   -
+     Outputs:  buffer:               the newly created iovec
+     PreCond:  -
+     PostCond: -
+     Perms:    -
+     Throws:   BadParameter
+               NoSuccess
+     Notes:    - all notes from the buffer CONSTRUCTOR apply.
+               - if len_in is larger than size, and size is
+                 not given as -1, a 'BadParameter' exception
+                 is thrown.
+
+   - DESTRUCTOR
+     Purpose:  destroy an iovec instance
+     Format:   DESTRUCTOR           (in  iovec obj);
+     Inputs:   obj:                  the iovec to destroy
+     InOuts:   -
+     Outputs:  -
+     PreCond:  -
+     PostCond: -
+     Perms:    -
+     Throws:   -
+     Notes:    - all notes from the buffer DESTRUCTOR apply.
+
+   - set_offset
+     Purpose:  set offset
+     Format:   set_offset           (in  int   offset);
+     Inputs:   offset:               value for offset
+     InOuts:   -
+     Outputs:  -
+     PreCond:  -
+     PostCond: -
+     Perms:    -
+     Throws:   BadParameter
+     Notes:    - if offset is smaller that zero, a
+                 'BadParameter' exception is thrown.
```

```
+
+   - get_offset
+     Purpose:  retrieve the current value for offset
+     Format:   get_offset           (out int   offset);
+     Inputs:   -
+     InOuts:   -
+     Outputs:  offset:               value of offset
+     PreCond:  -
+     PostCond: -
+     Perms:    -
+     Throws:   -
+     Notes:    -
+
+   - set_len_in
+     Purpose:  set len_in
+     Format:   set_len_in           (in  int   len_in);
+     Inputs:   len_in:               value for len_in
+     InOuts:   -
+     Outputs:  -
+     PreCond:  -
+     PostCond: -
+     Perms:    -
+     Throws:   BadParameter
+     Notes:    - if len_in is larger than size, and size is
+                 not set to -1, a 'BadParameter' exception
+                 is thrown.
+
+   - get_len_in
+     Purpose:  retrieve the current value for len_in
+     Format:   get_len_in           (out int   len_in);
+     Inputs:   -
+     InOuts:   -
+     Outputs:  len_in:               value of len_in
+     PreCond:  -
+     PostCond: -
+     Perms:    -
+     Throws:   -
+     Notes:    -
+
+   - get_len_out
+     Purpose:  retrieve the value for len_out
+     Format:   get_len_out          (out int   len_out);
+     Inputs:   -
+     InOuts:   -
+     Outputs:  len_out:              value of len_out
+     PreCond:  -
```

```
+     PostCond: -
+     Perms:    -
+     Throws:   -
+     Notes:    - len_out reports the number of bytes read
+                 or written in a completed read_w or write_w
+                 operation.
+               - before completion of the operation, the
+                 returned value is -1.
+               - for implementation managed memory, the
+                 value of len_out is always the same as
+                 for size.
```

Class `file`

This class represents an open file descriptor for read/write operations on a physical file. Its concept is similar to the file descriptor returned by the `open (2)` call in POSIX.

Several *In language bindings where this is appropriate, several* methods can return error codes indicating failure, instead of always raising an exception. These error codes are, as described in *the saga error section* *Section 3.1*, defined as POSIX `errno` values. These codes SHOULD be used in identical situations as described in POSIX. The calls which can use return error codes are documented.

```
    - CONSTRUCTOR
      Purpose:  create the obj
!     Format:   CONSTRUCTOR        (in  session   s,
                                    in  saga::url name,
                                    in  int       flags = Read,
                                    out file      obj)
!     Inputs:   s:                 session to associate the
!                                  object with
                name:              location of file
                flags:             mode for opening
+     InOuts:   -
      Outputs:  obj:               the newly created object
+     PreCond:  -
+     PostCond: - the file is opened.
+               - 'Owner' of target is the id of the context
+                 use to perform the opereration, if the file
+                 gets created.
+     Perms:    Exec  for parent directory.
```

```
+               Write for parent directory if Create is set.
+               Write for name if Write is set.
+               Read  for name if Read  is set.
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
                AlreadyExists
                DoesNotExist
-               IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - all notes from the directory::open() method
                  apply.
!               - the default flags are 'Read' (512).


    - DESTRUCTOR
      Purpose:  destroy the object
      Format:   DESTRUCTOR         (in  file      obj)
      Inputs:   obj:                the object to destroy
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - the file is closed.
+     Perms:    -
      Throws:   -
      Notes:    - the semantics of the inherited destructors
                  apply


    additional inspection methods:
    ------------------------------

    - get_size
      Purpose:  returns the number of bytes in the file
      Format:   get_size           (out int     size);
      Inputs:   -
+     InOuts:   -
      Outputs:  size:               number of bytes in the file
+     PreCond:  -
+     PostCond: -
+     Perms:    Query
      Throws:   NotImplemented
```

```
-               IncorrectURL
-               BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - similar to the 'st_size' field from 'stat' (2)
                  as defined by POSIX


    POSIX-like I/O methods:
    -----------------------

    - read
      Purpose:  reads up to len_in bytes from the file into
                the buffer.
-     Format:   read              (in    array<byte> buffer,
+     Format:   read              (inout buffer      buf,
!                                  in    int         len_in = -1,
                                   out   int         len_out);
      Inputs:   len_in:           number of bytes to be read
!     InOuts:   buf:              buffer to read data into
      Outputs:  len_out:          number of bytes successfully
                                  read
+     PreCond:  -
+     PostCond: - the data from the file are available in the
+                 buffer.
+     Perms:    Read
      Throws:   NotImplemented
                BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
!     Notes:    - the actual number of bytes read into buffer
                  is returned in len_out.  It is not an error
                  to read less bytes than requested, or in fact
                  zero bytes, e.g. at the end of the file.
                - errors are indicated by returning negative
                  values for len_out, which correspond to
!                 negatives of the respective POSIX ERRNO error
                  code.
```

```
                - the file pointer is positioned at the end of
                  the byte area successfully read during this
                  call.
!               - the given buffer must be large enough to
!                 store up to len_in bytes, or managed by the
!                 implementation - otherwise a 'BadParameter'
!                 exception is thrown.
+               - the notes about memory management from the
+                 buffer class apply.
!               - if the file was opened in write-only mode (i.e.
!                 no 'Read' or 'ReadWrite' flag was given), this
!                 method throws an 'PermissionDenied' exception.
!               - if len_in is smaller than 0, or not given,
!                 the buffer size is used for len_in.
!                 If that is also not available, a
!                 'BadParameter' exception is thrown.
                - similar to read (2) as specified by POSIX


    - write
      Purpose:  writes up to len_in bytes from buffer into
                the file at the current file position.
-     Format:   write              (in  array<byte> buffer,
+     Format:   write              (in  buffer      buf,
!                                   in  int         len_in = -1,
                                    out int         len_out);
      Inputs:   len_in:             number of bytes to write
!               buf:                buffer to write data from
+     InOuts:   -
      Outputs:  len_out:            number of bytes successfully
                                    written
+     PreCond:  -
+     PostCond: - the buffer data are written to the file.
+     Perms:    Write
      Throws:   NotImplemented
                BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - errors are indicated by returning negative
                  values for len_out, which correspond to
                  negatives of the respective POSIX ERRNO error
                  code.
```

```
                - the file pointer is positioned at the end
                  of the byte area written during this call.
!               - if the file was opened in read-only mode (i.e.
!                 no 'Write' or 'ReadWrite' flag was given), this
!                 method throws an 'PermissionDenied' exception.
+               - the given buffer must hold enough data to
+                 write - otherwise, only the available data
+                 will be written, and and len_out will be set
+                 to the number of bytes written.
+               - the notes about memory management from the
+                 buffer class apply.
!               - if len_in is smaller than 0, or not given,
!                 the buffer size is used for len_in.
!                 If that is also not available, a
!                 'BadParameter' exception is thrown.
                - if data are written beyond the current end of
                  file, the intermediate gap is filled with '\0'
                  bytes.
                - similar to write (2) as specified by POSIX


    - seek
      Purpose:  reposition the file pointer
      Format:   seek               (in  int        offset,
                                    in  seek_mode  whence,
                                    out int        position);
      Inputs:   offset:             offset in bytes to move
                                    pointer
                whence:             offset is relative to
                                    'whence'
+     InOuts:   -
      Outputs:  position:           position of pointer after
                                    seek
+     PreCond:  -
+     PostCond: - the file pointer is moved to the new position.
+               - following read() or write() operations use
+                 that position.
+     Perms:    Read or Write.
      Throws:   NotImplemented
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - seek repositions the file pointer for
```

```
                subsequent read, write and seek calls.
              - initially (after open), the file pointer is
                positioned at the beginning of the file,
                unless the 'Append' flag was given - then
                the initial position is the end of the file.
              - the repositioning is done relative to the
                position given in 'Whence', so relative to
                the 'Begin' or 'End' of the file, or to the
                'Current' position.
              - errors are indicated by returning negative
                values for len_out, which correspond to
                negatives of the respective POSIX ERRNO error
                code.
              - the file pointer can be positioned after the
!               end of the file without extending it.
              - the given offset can be positive, negative, or
                zero.
!             - note that a subsequent read at or behind the end
!               of file returns no data.
              - similar to lseek (2) as specified by POSIX.


    Scattered I/O methods:
    ----------------------

    - read_v
      Purpose:  gather/scatter read
!     Format:   read_v             (inout array<iovec> iovecs);
+     Inputs:   -
!     InOuts:   iovecs:             array of iovec structs
                                    defining start (offset) and
                                    length (len_in) of each
                                    individual read, the buffer
                                    to read into, and integer
                                    to store result into
                                    (len_out).
+     Outputs:  -
+     PreCond:  -
+     PostCond: - data from the file are available in the
+               iovec buffers.
+     Perms:    Read
      Throws:   NotImplemented
                BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
```

```
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - the behaviour of each individual read is as
                  in the normal read method, and all notes from
                  the read() method apply.
!               - an exception MUST be thrown if any of the
                  individual reads detects a condition which
                  would raise an exception for the normal
                  read() method.
+               - the notes about memory management from the
+                 buffer class apply.
!               - if for any of the given iovecs no len_in is
!                 given, then the buffer size is used as len_in.
!                 If that is also not available, a
!                 'BadParameter' exception is thrown.
+               - if the file was opened WriteOnly, a
+                 'PermissionDenied' exception is thrown.
                - similar to readv (2) as specified by POSIX


    - write_v
      Purpose:  gather/scatter write
!     Format:   write_v            (inout  array<iovec> iovecs);
+     Inputs:   -
!     InOuts:   iovecs:             array of iovec structs
                                    defining start (offset) and
                                    length (len_in) of each
                                    individual write, and
                                    buffers containing the data
                                    to write (len_out)
+     Outputs:  -
+     PreCond:  -
+     PostCond: - the iovec buffer data are written to the file.
+     Perms:    Write
      Throws:   NotImplemented
                IncorrectState
                BadParameter
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - the behaviour of each individual write is as
                  in the normal write method.
!               - an exception MUST be thrown if any of the
```

```
                individual writes detects a condition which
                would raise an exception for the normal write
                method.
+             - the notes about memory management from the
+               buffer class apply.
!             - if for any of the given iovecs no len_in is
!               given, then the buffer size is used as len_in.
!               If that is also not available, a
!               'BadParameter' exception is thrown.
+             - if the file was opened ReadOnly, a
+               'PermissionDenied' exception is thrown.
              - similar to writev (2) as specified by POSIX


    Pattern-based I/O methods:
    --------------------------

    - size_p
!     Purpose:  determine the storage size required for a
                pattern I/O operation
      Format:   size_p             (in  string  pattern,
                                    out int     size);
      Inputs:   pattern:            pattern to determine size for
+     InOuts:   -
      Outputs:  size:               size required for I/O
                                    operation with that pattern
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   NotImplemented
                BadParameter
+               IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - the method does, in general, not perform a
                  remote operation, but is intended to help
                  the application programmer to correctly handle
                  pattern-based I/O and associated buffer sizes.
                - if the pattern cannot be parsed or interpreted,
                  a 'BadParameter' exception is thrown.


    - read_p
```

```
      Purpose:  pattern-based read
      Format:   read_p             (in    string      pattern,
-                                   inout array<byte> buffer,
+                                   inout buffer      buf,
                                    out   int         len_out);
      Inputs:   pattern:            pattern specification for
                                    read operation
!     InOuts:   buf:                buffer to store read data
                                    into
      Outputs:  len_out:            number of successfully read
                                    bytes
+     PreCond:  -
+     PostCond: - data from the file are available in the
+                 buffers.
+     Perms:    Read
      Throws:   NotImplemented
                BadParameter
-               AlreadyExists
-               DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
-     Throws:   BadParameter
      Notes:    - if the pattern cannot be parsed or interpreted,
                  a 'BadParameter' exception is thrown.
                - all notes for the read() method apply for the
                  individual reads resulting from the
                  interpretation of the pattern.
+               - an exception MUST be thrown if any of the
+                 individual writes detects a condition which
+                 would raise an exception for the normal write
+                 method.


    - write_p
      Purpose:  pattern-based read
!     Format:   write_p            (in    string      pattern,
-                                   in    array<byte> buffer,
+                                   in    buffer      buf,
                                    out   int         len_out);
      Inputs:   pattern:            pattern specification for
!                                   write operation
!               buf:                buffer to be written
```

```
+     InOuts:   -
      Outputs:  len_out:            number of bytes successfully
                                    written
+     PreCond:  -
+     PostCond: - the buffer data are written to the file.
+     Perms:    Write
      Throws:   NotImplemented
                BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - if the pattern cannot be parsed or interpreted,
                  a 'BadParameter' exception is thrown.
                - all notes for the write() method apply for the
                  individual writes resulting from the
                  interpretation of the pattern.
+               - an exception MUST be thrown if any of the
+                 individual writes detects a condition which
+                 would raise an exception for the normal write
+                 method.


    Extended I/O methods:
    ---------------------

    - modes_e
!     Purpose:  list the exetnded modes available in this
                implementation, and/or on server side
      Format:   modes_e            (out array<string> emodes);
      Inputs:   -
+     InOuts:   -
      Outputs:  emodes:             list of modes available for
                                    extended I/O
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   NotImplemented
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
```

```
      Notes:    - the method does, in general, not perform a
                  remote operation, but is intended to help
                  the application programmer to determine what
                  extended I/O methods are supported by the
                  implementation.


+   - size_e
+     Purpose:  determine the strorage size required for an
+               extended I/O operation
+     Format:   size_e             (in  string  emode,
+                                  (in  string  spec,
+                                   out int     size);
+     Inputs:   emode:              extended mode to use
+               spec:               specification to determine
+                                   size for
+     InOuts:   -
+     Outputs:  size:               size required for I/O
+                                   operation with that
+                                   emode/spec
+     PreCond:  -
+     PostCond: -
+     Perms:    -
+     Throws:   NotImplemented
+               BadParameter
+               IncorrectState
+               PermissionDenied
+               AuthorizationFailed
+               AuthenticationFailed
+               Timeout
+               NoSuccess
+     Notes:    - the method does, in general, not perform a
+                 remote operation, but is intended to help
+                 the application programmer to correctly handle
+                 extended I/O and associated buffer sizes.
+               - if the specification cannot be parsed or
+                 interpreted, a 'BadParameter' exception is
+                 thrown.


    - read_e
      Purpose:  extended read
      Format:   read_e             (in    string      emode,
                                    in    string      spec,
-                                   inout array<byte> buffer,
+                                   inout buffer      buf,
```

```
                                    out   int         len_out);
      Inputs:   emode:              extended mode to use
                spec:               specification of read
                                    operation
!     InOuts:   buf:                buffer to store read data
                                    into
      Outputs:  len_out:            number of successfully read
                                    bytes
+     PreCond:  -
+     PostCond: - data from the file are available in the
+               buffers.
+     Perms:    Read
      Throws:   NotImplemented
                BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - if the emode is not supported, a 'BadParameter'
                  exception is thrown.
                - if the spec cannot be parsed or interpreted,
                  a 'BadParameter' exception is thrown.
                - all notes from the read() method apply to the
                  individual reads resulting from the
!                 interpretation of the emode and spec.
+               - an exception MUST be thrown if any of the
+                 individual writes detects a condition which
+                 would raise an exception for the normal write
+                 method.


    - write_e
      Purpose:  extended write
      Format:   write_e            (in    string      emode,
                                    in    string      spec,
-                                   in    array<byte> buffer,
+                                   in    buffer      buf,
                                    out   int         len_out);
      Inputs:   emode:              extended mode to use
                spec:               specification of write
                                    operation
!               buf:                buffer to store read data
                                    into
+     InOuts:   -
```

```
!     Outputs:  len_out:            number of bytes successfully
!                                   written
+     PreCond:  -
+     PostCond: - the buffer data are written to the file.
+     Perms:    Write
      Throws:   NotImplemented
                BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - if the emode is not supported, a 'BadParameter'
                  exception is thrown.
                - if the spec cannot be parsed or interpreted,
                  a 'BadParameter' exception is thrown.
                - all notes from the write() method apply to the
                  individual writes resulting from the
                  interpretation of the 'emode' and 'spec'.
+               - an exception MUST be thrown if any of the
+                 individual writes detects a condition which
+                 would raise an exception for the normal write
+                 method.
```

Class `directory`

```
    - CONSTRUCTOR
      Purpose:  open the directory
!     Format:   CONSTRUCTOR        (in  session    s,
                                    in  saga::url  name,
                                    in  int        flags = Read,
!                                   out directory  obj)
!     Inputs:   s:                  session to associate the
                                    object with
                name:               location of directory
                flags:              mode for opening
+     InOuts:   -
!     Outputs:  obj:                the newly created object
+     PreCond:  -
+     PostCond: - the directory is opened.
+               - 'Owner' of target is the id of the context
```

```
+                 use to perform the opereration, if the
+                 directory gets created.
+     Perms:      Exec  for parent directory.
+                 Write for parent directory if Create is set.
+                 Write for name if Write is set.
+                 Read  for name if Read  is set.
      Throws:     NotImplemented
                  IncorrectURL
                  BadParameter
                  AlreadyExists
                  DoesNotExist
                  PermissionDenied
                  AuthorizationFailed
                  AuthenticationFailed
                  Timeout
                  NoSuccess
!     Notes:      - the default flags are 'Read' (512).
                  - the semantics of the inherited constructors
                    apply


    - DESTRUCTOR
      Purpose:  destroy the directory object
!     Format:   DESTRUCTOR         (in  directory obj)
!     Inputs:   obj:                the object to destroy
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - the directory is closed.
+     Perms:    -
      Throws:   -
      Notes:    - the semantics of the inherited destructors
                  apply.


    inspection methods:
    -------------------

    - get_size
      Purpose:  returns the number of bytes in the file
      Format:   get_size           (in  saga::url name,
                                    in  int       flags = None,
                                    out int       size);
      Inputs:   name:               name of file to inspect
                flags:              mode for operation
+     InOuts:   -
```

```
      Outputs:  size:               number of bytes in the file
+     PreCond:  -
+     PostCond: -
      Perms:    Query
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - if 'name' can be parsed as URL, but contains
                  an invalid entry name, a 'BadParameter'
                  exception is thrown.
                - if the entry 'name' points to does not exist,
                  a 'DoesNotExist' exception is thrown.
                - if the 'name' points to a link and the
                  'Dereference' flag is set, the size is
                  returned for the link target.  If that target
                  does not exist, a 'DoesNotExist' exception is
                  thrown.
                - the default flags are 'None' (0).
                - other flags are not allowed on this method,
                  and cause a 'BadParameter' exception.
                - similar to the 'st_size' field from 'stat' (2)
                  as defined by POSIX

    - is_file
      Alias:    for is_entry in saga::ns_directory


    Factory-like methods for creating objects:
    ------------------------------------------


    - open_dir
      Purpose:  creates a directory object
      Format:   open_dir           (in  saga::url name,
                                    in  int       flags = Read,
                                    out directory dir)
      Inputs:   name:               name of directory to open
!               flags:              flags defining operation
                                    modus
+     InOuts:   -
```

```
      Outputs:  dir:                opened directory instance
+     PreCond:  -
+     PostCond: - the session of the returned instance is that of
+                 the calling instance.
+               - 'Owner' of name is the id of the context
+                 used to perform the opereration if name gets
+                 created.
+     Perms:    Exec  for name's parent directory.
+               Write for name's parent directory if Create is set.
+               Write for name if Write is set.
+               Read  for name if Read  is set.
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
                AlreadyExists
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - all notes from the ns_directory::open_dir()
                  method apply.
!               - default flags are 'Read' (512).


    - open
      Purpose:  creates a new file instance
      Format:   open               (in  saga::url name,
                                    in  int       flags = Read,
                                    out file      file);
      Inputs:   name:               file to be opened
!               flags:              flags defining operation
                                    modus
+     InOuts:   -
      Outputs:  file:               opened file instance
+     PreCond:  -
+     PostCond: - the session of the returned instance is that of
+                 the calling instance.
+               - 'Owner' of name is the id of the context
+                 used to perform the opereration if name gets
+                 created.
+     Perms:    Exec  for name's parent directory.
+               Write for name's parent directory if Create is set.
+               Write for name if Write is set.
```

```
+               Read  for name if Read  is set.
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
                AlreadyExists
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - all notes from the ns_directory::open() method
                  apply.
                - the file is truncated to length 0 on the open
                  operation if the 'Trunc' flag is given.
                - the file is in opened in append mode if the
                  'Append' flag is given (a seek(0, End) is
                  performed after the open).  If the 'Append'
                  flag is not given, the file pointer is
!                 initially placed at the beginning of the file
                  (a seek(0,Start) is performed after the open).
                - the 'Binary' flag is to be silently ignored on
!                 systems which do not support it.
+               - at least one of the flags 'Read', 'Write' or
+                 'ReadWrite' must be given, otherwise a
+                 'BadParameter' exception is thrown.
                - the flag set 'Read | Write' is equivalent to
                  the flag 'ReadWrite'.
!               - default flags are 'Read' (512).
```

4.3.3 Examples

Example: open a file. If its size is greater than 10, then read the first 10 bytes into a string, and print it.

Code Example

```
1   // c++ example
2   void head (saga::url url)
3   {
4     try {
5       // get type and other infos
6       saga::file f (url);
7
```

```
8       off_t size = f.get_size ();
9
10      if ( size > 10 )
11      {
12        char   buf[11];
13
14        ssize_t len_out = f.read (saga::buffer (buf));
15
16        if ( 10 == len_out )
17        {
18          std::cout << "head: "
19                    << buffer.get_data ()
20                    << std::endl;
21        }
22      }
23    }
24
25    // catch any possible error - see elsewhere for better
26    // examples of error handling in SAGA
27    catch ( const saga::exception & e )
28    {
29      std::cerr << "Oops! SAGA error: "
30                << e.get_message ()
31                << std::endl;
32    }
33
34    return;
35  }
```

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.

The `saga::logical_file` class implements the `saga::attributes` interface. It is important to realize that this is **intended** to reflect the ability of replica systems to associate meta data with logical files. The SAGA attribute model (string based key/value pairs) can, with all **probability**, only give a crude representation of meta data models used in real world replica systems – however, the definition of a more abstract and comprehensive data model for replica meta data was felt to be outside the scope of a SAGA API definition. Implementations are expected to map the native data model to key/value pairs as well as possible, and MUST document that mapping process (and in particular the supported keys) carefully.

Please note that the interactions with logical files as opaque entities (as entries in logical file name spaces) are covered by the `namespace` package. The interfaces presented here supplement the `namespace` package with operations for operating on entries in replica catalogues.

It is up to the used backend to ensure that multiple replica locations registered on a logical file are indeed identical copies – the SAGA API does not imply any specific consistency model. The SAGA implementation MUST document the consistency model used.

4.4.1 Definitions

Logical File: A *logical file* represents merely an entry in a name space which has (a) an associated set of registered (physical) replicas of that file, and (b) an associated set of meta data describing that logical file. Both sets can be empty. *To access the content of a logical file, a `saga::file` needs to be created with one of the registered replica locations.*

Replica: A *replica* (or *physical file*) is a file which is registered on a logical file. In general, all replicas registered on the same logical *file* are identical. Often, one of these replicas is deemed to be a master **copy** (often **it is** the first replica registered, and/or the only one which can be changed) – that distinction is, however, not visible in the SAGA API.

Logical Directory: A *logical directory* represents a directory entry in the name space of logical files. Several replica system implementations have the notion of *container*s, which, for our purposes, represent directories which can have, just as logical files, associated sets of meta data. In the presented API, logical directories and containers are the same.

Note that the `Truncate`, `Append` and `Binary` flags have no meaning on logical files. The respective enum values for these flags for `saga::file`s have been reserved though, for (a) future use, and (b) consistency with the `saga::file` flag values.

The `find()` method of the `saga::logical_directory` class represents a combination of (a) the `find()` method from the `saga::ns_directory` class, and (b) the `find_attributes()` method from the `saga::attributes` interface. The method accepts patterns for meta data matches (*key_pattern and val_pattern* ***attr_pattern***) and *a single pattern* for file name matches (`name_pattern`), and returns a list of logical file names *for which both patterns match which match all* ***attr_pattern*** *and the* ***name_pattern*** *(AND semantics)*. The ***attr_pattern*** are formatted as defined for `find_attribute()` of the `saga::attributes` interface. The `name_pattern` are formatted as defined for the `find()` method of the `saga::ns_directory` class. In general, the allowed patterns are the same as defined as wildcards in the **description** of the SAGA `namespace` package.

4.4.2 Specification

```
  package saga.logical_file
  {
    enum flags
    {
!     None            =    0, // same as in namespace::flags
!     Overwrite       =    1, // same as in namespace::flags
!     Recursive       =    2, // same as in namespace::flags
!     Dereference     =    4, // same as in namespace::flags
!     Create          =    8, // same as in namespace::flags
!     Exclusive       =   16, // same as in namespace::flags
!     Lock            =   32, // same as in namespace::flags
!     CreateParents   =   64, // same as in namespace::flags
      //                 128,    reserved for Truncate
      //                 256,    reserved for Append
      Read            =  512,
      Write           = 1024,
      ReadWrite       = 1536, // Read | Write
      //                2048     reserved for Binary
```

```
    }


    class logical_file : extends        saga::ns_entry
                         implements     saga::attributes
                       // from ns_entry  saga::object
                       // from ns_entry  saga::async
                       // from object    saga::error_handler
    {
!     CONSTRUCTOR     (in  session             s,
                        in  saga::url           name,
                        in  int                 flags = Read,
!                       out logical_file        obj);
!     DESTRUCTOR      (in  logical_file        obj);


      // manage the set of associated replicas
      add_location    (in  saga::url           name);
      remove_location (in  saga::url           name);
      update_location (in  saga::url           name_old,
                       in  saga::url           name_new);
      list_locations  (out array<saga::url>    names);

      // create a new physical replica
      replicate       (in  saga::url           name,
                       in  int                 flags = None);

      // Attributes (extensible):
+     //
+     // no attributes pre-defined
    }


    class logical_directory : extends            saga::ns_directory
                              implements         saga::attributes
                            // from ns_directory  saga::ns_entry
                            // from ns_entry      saga::object
                            // from ns_entry      saga::async
                            // from object        saga::error_handler
    {

!     CONSTRUCTOR     (in  session             s,
                        in  saga::url           name,
                        in  int                 flags = Read,
!                       out logical_directory   obj);
!     DESTRUCTOR      (in  logical_directory   obj);
```

```
!     // inspection methods
      is_file         (in  saga::url           name,
-                      in  int                 flags = None,
                       out boolean             test);

      // open methods
      open_dir        (in  saga::url           name,
                       in  int                 flags = Read,
                       out logical_directory   dir);

      open            (in  saga::url           name,
                       in  int                 flags = Read,
                       out logical_file        file);

      // find logical files based on name and meta data
      find            (in  string              name_pattern,
!                      in  array<string>       attr_pattern,
-                      in  array<string>       val_pattern,
-                      in  int                 flags = None,
+                      in  int                 flags = Recursive,
                       out array<saga::url>    names   );
    }
  }
```

4.4.3 Specification Details

Enum `flags`

The ***flags*** *enum is inherited from the* ***namespace*** *package. A number of replica specific flags are added to it. All added flags are used for the opening of* ***logical_file*** *and* ***logical_directory*** *instances, and are not applicable to the operations inherited from the* ***namespace*** *package.*

`Read`
: *The logical file or directory is opened for reading – that does not imply the ability to change the logical file or directory.*

`Write`
: *The logical file or directory is opened for writing – that does not imply the ability to read from the logical file or directory.*

`ReadWrite`

The logical file or directory is opened for reading and writing.

Class `logical_file`

This class provides *the* means to handle the contents of logical files. These contents consists of strings representing locations of physical files (replicas) associated with the logical file.

```
    - CONSTRUCTOR
      Purpose:  create the object
!     Format:   CONSTRUCTOR      (in session    s,
                                  in  saga::url name,
                                  in  int       flags = Read,
!                                 out logical_file  obj)
!     Inputs:   s:                session to associate with
!                                 the object
                name:             location of file
                flags:            mode for opening
+     InOuts:   -
      Outputs:  obj:              the newly created object
+     PreCond:  -
+     PostCond: - the logical_file is opened.
+               - 'Owner' of target is the id of the context
+                 use to perform the opereration, if the
+                 logical_file gets created.
+     Perms:    Exec  for parent directory.
+               Write for parent directory if Create is set.
+               Write for name if Write is set.
+               Read  for name if Read  is set.
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
                AlreadyExists
                DoesNotExist
-               IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - the semantics of the inherited constructors
!                 and of the logical_directory::open() method
                  apply.
```

```
!               - the default flags are 'Read' (512).


    - DESTRUCTOR
      Purpose:  destroy the object
      Format:   DESTRUCTOR       (in  logical_file   obj)
      Inputs:   obj:              the object to destroy
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - the logical_file is closed.
+     Perms:    -
+     Throws:   -
      Notes:    - the semantics of the inherited destructors
                  apply.


      manage the set of associated replicas:
      --------------------------------------

    - add_location
      Purpose:  add a replica location to the replica set
      Format:   add_location     (in  saga::url name);
      Inputs:   name:             location to add to set
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - name is in the list of replica locations for
+                 the logical file.
+     Perms:    Write
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
-               ReadOnly
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - this methods adds a given replica location
                  (name) to the set of locations associated with
                  the logical file.
                - the implementation MAY choose to interpret the
                  replica locations associated with the logical
!                 file.  It MAY return an 'IncorrectURL' error
```

```
                  indicating an invalid location if it is unable
!                 or unwilling to handle that specific locations
!                 scheme.  The implementation documentation MUST
!                 specify how valid replica locations are formed.
                - if 'name' can be parsed as URL, but contains
!                 an invalid entry name, a 'BadParameter'
                  exception is thrown.
                - if the replica is already in the set, this
!                 method does nothing, and in particular MUST
!                 NOT raise an 'AlreadyExists' exception
+               - if the logical file was opened ReadOnly, a
+                 'PermissionDenied' exception is thrown.


    - remove_location
!     Purpose:  remove a replica location from the replica set
      Format:   remove_location  (in  saga::url name);
      Inputs:   name:             replica to remove from set
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - name is not anymore in list of replica
+                 locations for the logical file.
+     Perms:    Write
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
                DoesNotExist
-               ReadOnly
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - this method removes a given replica location
                  from the set of replicas associated with the
                  logical file.
                - the implementation MAY choose to interpret the
                  replica locations associated with the logical
!                 file.  It MAY return an 'IncorrectURL' error
                  indicating an invalid location if it is unable
!                 or unwilling to handle that specific locations
!                 scheme.  The implementation documentation MUST
!                 specify how valid replica locations are formed.
                - if 'name' can be parsed as URL, but contains
```

```
!                 an invalid entry name, a 'BadParameter'
                  exception is thrown.
                - if the location is not in the set of
                  replicas, a 'DoesNotExist' exception is
                  thrown.
                - if the set of locations is empty after this
                  operation, the logical file object is still
                  a valid object (see replicate() method
                  description).
+               - if the logical file was opened ReadOnly, a
+                 'PermissionDenied' exception is thrown.


    - update_location
      Purpose:  change a replica location in replica set
      Format:   update_location  (in saga::url name_old,
                                  in saga::url name_new);
      Inputs:   name_old          replica to be updated
!               name_new          update of replica
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - name_old is not anymore in list of replica
+               locations for the logical file.
+               - name_new is in the list of replica locations
+               for the logical file.
+     Perms:    Read
+               Write
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
                AlreadyExists
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - this method removes a given replica location
                  from the set of locations associated with the
                  logical file, and adds a new location.
                - the implementation MAY choose to interpret the
                  replica locations associated with the logical
!                 file.  It MAY return an 'IncorrectURL' error
                  indicating an invalid location if it is unable
```

```
!                 or unwilling to handle that specific locations
!                 scheme.  The implementation documentation MUST
!                 specify how valid replica locations are formed.
                - if 'name' can be parsed as URL, but contains
!                 an invalid entry name, a 'BadParameter'
                  exception is thrown.
                - if the old replica location is not in the
!                 set of locations, a 'DoesNotExist' exception
                  is thrown.
-                 and the new replica location is not added.
+               - if the new replica location is already in the
+                 set of locations, an 'AlreadyExists' exception
+                 is thrown.
+               - if the logical file was opened ReadOnly, an
+                 'PermissionDenied' exception is thrown.
+               - if the logical file was opened WriteOnly, an
+                 'PermissionDenied' exception is thrown.


    - list_locations
      Purpose:  list the locations in the location set
      Format:   list_locations   (out array<saga::url> names);
      Inputs:   -
+     InOuts:   -
      Outputs:  names:            array of locations in set
+     PreCond:  -
+     PostCond: -
+     Perms:    Read
      Throws:   NotImplemented
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - this method returns an array of urls
                  containing the complete set of locations
                  associated with the logical file.
                - an empty array returned is not an error -
                  the logical file object is still a valid
                  object (see replicate() method description).
+               - if the logical file was opened WriteOnly, an
+                 'PermissionDenied' exception is thrown.


    - replicate
```

```
      Purpose:  replicate a file from any of the known
                replica locations to a new location, and, on
                success, add the new replica location to the
                set of associated replicas
      Format:   replicate        (in  saga::url name,
                                  in  int       flags = None);
      Inputs:   name:             location to replicate to
                flags:            flags defining the operation
                                  modus
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - an identical copy of one of the available
+                 replicas exists at name.
+               - name is in the list of replica locations
+                 for the logical file.
+     Perms:    Read
+               Write
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
                AlreadyExists
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - the method implies a two step operation:
!                 1) create a new and comlete replica at the
!                    given location, which then represents
                     a new replica location.
                  2) perform an add_location() for the new
                     replica location.
                - all notes to the saga::ns_entry::copy() and
                  saga::locaical_file::add_location methods
                  apply.
                - the method is not required to be atomic, but:
                  the implementation MUST be either
                  successfull in both steps, or throw an
                  exception indicating if both methods failed,
                  or if one of the methods succeeded.
                - a replicate call on an instance with empty
!                 location set raises an 'IncorrectState'
                  exception, with an descriptive error message.
```

```
!               - the default flags are 'None' (0).  The
                  interpretation of flags is as described for
                  the ns_entry::copy() method.
+               - The 'Recursive' flag is not allowed, and
+                 causes a 'BadParameter' exception.
+               - if the logical file was opened ReadOnly, an
+                 'PermissionDenied' exception is thrown.
+               - if the logical file was opened WriteOnly, an
+                 'PermissionDenied' exception is thrown.
```

Class `logical_directory`

This class represents a container for logical files in a logical file name space. It allows traversal of the catalog's name space, and the manipulation and creation (open) of logical files in that name space.

```
    Constructor / Destructor:
    -------------------------

    - CONSTRUCTOR
      Purpose:  create the object
!     Format:   CONSTRUCTOR      (in  session       s,
                                  in  saga::url     name,
                                  in  int           flags = Read,
                                  out logical_directory
                                                    obj)
!     Inputs:   s:               session to associate with
                                  the object
                name:             location of directory
                flags:            mode for opening
+     InOuts:   -
      Outputs:  obj:              the newly created object
+     PreCond:  -
+     PostCond: - the logical_directory is opened.
+               - 'Owner' of target is the id of the context
+                 use to perform the opereration, if the
+                 logical_directory gets created.
+     Perms:    Exec  for parent directory.
+               Write for parent directory if Create is set.
+               Write for name if Write is set.
+               Read  for name if Read  is set.
```

```
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
+               AlreadyExists
                DoesNotExist
                PermissionDenied
                AuthorizationFailed
!               AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - the semantics of the inherited constructors
                  and of the logical_directory::open_dir()
                  method apply.
!               - the default flags are 'Read' (512).


    - DESTRUCTOR
      Purpose:  destroy the object
      Format:   DESTRUCTOR         (in  logical_directory obj)
      Inputs:   obj:                the object to destroy
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - the logical_directory is closed.
+     Perms:    -
+     Throws:   -
      Notes:    - the semantics of the inherited destructors
                  apply.


    - is_file
      Alias:    for is_entry of saga::ns_directory


    - open_dir
      Purpose:  creates a new logical_directory instance
      Format:   open_dir         (in  saga::url name,
                                  in  int       flags = Read,
                                  out logical_directory dir);
      Inputs:   name:             name of directory to open
!               flags:            flags defining operation
                                  modus
+     InOuts:   -
      Outputs:  dir:              opened directory instance
+     PreCond:  -
+     PostCond: - the session of the returned instance is that of
```

```
+                the calling instance.
+              - 'Owner' of name is the id of the context
+                used to perform the opereration if name gets
+                created.
+     Perms:   Exec  for name's parent directory.
+              Write for name's parent directory if Create is set.
+              Write for name if Write is set.
+              Read  for name if Read  is set.
      Throws:  NotImplemented
               IncorrectURL
               BadParameter
               AlreadyExists
               DoesNotExist
               IncorrectState
               PermissionDenied
               AuthorizationFailed
               AuthenticationFailed
               Timeout
               NoSuccess
      Notes:   - all notes from the ns_directory::open_dir()
                 method apply.
!             - default flags are 'Read' (512).


    - open
      Purpose: creates a new logical_file instance
      Format:  open             (in  saga::url    name,
                                 in  int          flags = Read,
                                 out logical_file file);
      Inputs:  name:             file to be opened
!             flags:             flags defining operation
                                 modus
+     InOuts:  -
      Outputs: file:             opened file instance
+     PreCond: -
+     PostCond: - the session of the returned instance is that of
+                 the calling instance.
+               - 'Owner' of name is the id of the context
+                 used to perform the opereration if name gets
+                 created.
+     Perms:   Exec  for name's parent directory.
+              Write for name's parent directory if Create is set.
+              Write for name if Write is set.
+              Read  for name if Read  is set.
      Throws:  NotImplemented
               IncorrectURL
```

```
                BadParameter
                AlreadyExists
                DoesNotExist
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - all notes from the ns_directory::open() method
                  apply.
                - the flag set 'Read | Write' is equivalent to
                  the flag 'ReadWrite'.
!               - default flags are 'Read' (512).


    - find
      Purpose:  find entries in the current directory and below,
                with matching names and matching meta data
      Format:   find             (in  string           name_pattern,
!                                 in  array<string>    attr_pattern,
-                                 in  array<string>    val_pattern,
-                                 in  int              flags = None,
+                                 in  int              flags = Recursive,
                                  out array<saga::url> names);
      Inputs:   name_pattern:     pattern for names of
                                  entries to be found
!               attr_pattern:     pattern for meta data
!                                 key/values of entries to be
                                  found
-               val_pattern:      pattern for meta data values
-                                 of entries to be found
                flags:            flags defining the operation
                                  modus
+     InOuts:   -
      Outputs:  names:            array of names matching both
                                  pattern
+     PreCond:  -
+     PostCond: -
+     Perms:    Read  for cwd.
                Query for entries specified by name_pattern.
+               Exec  for parent directories of these entries.
+               Query for parent directories of these entries.
+               Read  for directories specified by name_pattern.
+               Exec  for directories specified by name_pattern.
+               Exec  for parent directories of these directories.
```

```
+               Query for parent directories of these directories.
      Throws:   NotImplemented
                BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
!     Notes:    - the description of find() in the Introduction
                  to this section applies.
                - the semantics for both the find_attributes()
                  method in the saga::attributes interface and
                  for the find() method in the
                  saga::ns_directory class apply.  On
!                 conflicts, the find() semantic supersedes
!                 the find_attributes() semantic.  Only entries
!                 matching all attribute patterns and the name
!                 space pattern are returned.
-               - the default flag set is 'None' (2).
+               - the default flags are 'Recursive' (2).
```

4.4.4 Examples

Code Example

```
1   // c++ example
2   int main ()
3   {
4     saga::logical_file lf ("lfn://remote.catalog.net/tmp/file1");
5
6     lf.replicate ("gsiftp://localhost//tmp/file.rep");
7     saga::file f ("gsiftp://localhost//tmp/file.rep");
8
9     std::cout << "size of local replica: "
10              << f.get_size ()
11              << std::endl;
12
13    return (0);
14  }
```

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 and encryption schemes*. The stream API *has the following characteristics*

1. *It* is not performance oriented: If performance is required, then it is better to program directly against the APIs of existing performance oriented protocols like GridFTP or XIO. The API design should allow, however, for *high* performance implementations.
2. *It* is focused on TCP/IP socket connections. There has been no attempt to generalize this to arbitrary streaming interfaces (although it does not prevent such things as connectionless **protocols** from being supported).
3. *It* does not attempt to create a programming paradigm that diverges very far from baseline BSD sockets, Winsock, or Java Sockets.

This API greatly reduces the complexity of establishing authenticated socket connections in order to communicate with remotely located components. It however, provides very limited functionality and is thus suitable for applications that do not have very sophisticated requirements (as per 80-20 rule). It is envisaged that as applications become progressively more sophisticated, they will *gradually move* to more *the* sophisticated, native APIs in order to support those needs.

Several SAGA use cases require a more abstract communication API, which exchanges opaque messages instead of byte streams. That behaviour can be modelled on top of this stream API, but future versions of the SAGA API may introduce higher level communication APIs.

4.5.1 Endpoint URLs

The SAGA stream API uses URLs to specify connection endpoints. These URLs are supposed to allow SAGA implementations to be interoperable. For example, the URL

```
tcp://remote.host.net:1234/
```

is supposed to signal that a standard `tcp` connection can be established with host `remote.host.net` on port `1234`. No matter what the specified URL scheme is, the SAGA stream API impementation MUST have the same semantics on API level, i.e. behave like a reliable byte-oriented data stream.

4.5.2 Endpoint Permissions

The SAGA API allows for application level authorization of stream communications: an application is able to set permissions on ***saga::stream_server*** *and* ***saga::stream*** *instances. These permissions control what remote party can perform what action on those streams, e.g. control what remote parties are able to connect to an endpoint, or to write to them etc.*

Not all implementations will be able to fully implement that security model – the implementation MUST carefully document which permissions are supported, and which are not.

4.5.3 Specification

```
  package saga.stream
  {
    enum state
    {
      New          =  1
      Open         =  2,
      Closed       =  3,
      Dropped      =  4,
      Error        =  5
    }

    enum activity
    {
      Read         =  1,
      Write        =  2,
      Exception    =  4
    }


    class stream_service : implements   saga::object
                           implements   saga::async
                           implements   saga::monitorable
+                          implements   saga::permissions
```

```
                        // from object  saga::error_handler
    {
!     CONSTRUCTOR       (in    session         s,
!                        in    saga::url       url,
                         out   stream_service  obj);
      DESTRUCTOR        (in    stream_service  obj);

!     get_url           (out   saga::url       url);

      serve             (in    float           timeout = -1.0,
                         out   stream          stream);

      close             (in    float           timeout = 0.0);

      // Metrics:
!     //   name:  stream_server.client_connect
      //   desc:  fires if a client connects
      //   mode:  ReadOnly
      //   unit:  1
      //   type:  Trigger
!     //   value: 1
    }


    class stream : extends      saga::object
                   implements   saga::async
                   implements   saga::attributes
                   implements   saga::monitorable
                // from object  saga::error_handler
    {
      // constructor / destructor
!     CONSTRUCTOR  (in    session          s,
!                   in    saga::url        url = "",
                    out   stream           obj);
      DESTRUCTOR   (in    stream           obj);

!     // inspection methods
!     get_url      (out   saga::url        url);
      get_context  (out   context          ctx);

!     // management methods
      connect      (void);
      wait         (in    int              what,
!                   in    float            timeout = -1.0,
!                   out   int              cause);
      close        (in    float            timeout = 0.0);
```

```
      // I/O methods
-     read         (inout array<byte>      buffer,
+     read         (inout buffer           buf,
!                   in    int             len_in = -1,
                    out   int             len_out);
-     write        (in    array<byte>      buffer,
+     write        (in    buffer           buf,
!                   in    int             len_in = -1,
                    out   int             len_out);

      // Attributes:
      //   name:  Bufsize
      //   desc:  determines the size of the send buffer,
      //          in bytes
      //   mode:  ReadWrite, optional
      //   type:  Int
      //   value: system dependend
      //   notes: - the implementation MUST document the
      //            default value, and its meaning (e.g. on what
      //            layer that buffer is maintained, or if it
!     //            disables zero copy).
      //
      //   name:  Timeout
      //   desc:  determines the amount of idle time
      //          before dropping the line, in seconds
      //   mode:  ReadWrite, optional
      //   type:  Int
      //   value: system dependend
      //   notes: - the implementation MUST document the
      //            default value
!     //          - if this attribute is supported, the
      //            connection MUST be closed by the
      //            implementation if for that many seconds
      //            nothing has been read from or written to
      //            the stream.
      //
      //   name:  Blocking
      //   desc:  determines if read/writes are blocking
      //          or not
      //   mode:  ReadWrite, optional
      //   type:  Bool
      //   value: True
      //   notes: - if the attribute is not supported, the
      //            implementation MUST be blocking
      //          - if the attribute is set to 'True', a read or
```

```
      //            write operation MAY return immediately if
!     //            no data can be read or written - that does
      //            not constitute an error (see EAGAIN in
      //            POSIX).
      //
      //   name:  Compression
      //   desc:  determines if data are compressed
      //          before/after transfer
      //   mode:  ReadWrite, optional
      //   type:  Bool
!     //   value: schema dependent
      //   notes: - the implementation MUST document the
      //            default values for the available schemas
      //
      //   name:  Nodelay
      //   desc:  determines if packets are sent
!     //          immediately, i.e. without delay
      //   mode:  ReadWrite, optional
      //   type:  Bool
      //   value: True
      //   notes: - similar to the TCP_NODELAY option
      //
      //   name:  Reliable
      //   desc:  determines if all sent data MUST arrive
      //   mode:  ReadWrite, optional
      //   type:  Bool
      //   value: True
      //   notes: - if the attribute is not supported, the
      //            implementation MUST be reliable


      // Metrics:
!     //   name:  stream.state
      //   desc:  fires if the state of the stream changes,
      //          and has the value of the new state
      //          enum
      //   mode:  ReadOnly
      //   unit:  1
      //   type:  Enum
!     //   value: New
      //
!     //   name:  stream.read
      //   desc:  fires if a stream gets readable
      //   mode:  ReadOnly
      //   unit:  1
      //   type:  Trigger
```

```
!     //   value: 1
      //   notes: - a stream is considered readable if a
      //             subsequent read() can sucessfully read
!     //             1 or more bytes of data.
      //
!     //   name:  stream.write
      //   desc:  fires if a stream gets writable
      //   mode:  ReadOnly
      //   unit:  1
      //   type:  Trigger
!     //   value: 1
      //   notes: - a stream is considered writable if a
      //             subsequent write() can sucessfully write
!     //             1 or more bytes of data.
      //
!     //   name:  stream.exception
      //   desc:  fires if a stream has an error condition
      //   mode:  ReadOnly
      //   unit:  1
      //   type:  Trigger
!     //   value: 1
      //   notes: -
      //
!     //   name:  stream.dropped
      //   desc:  fires if the stream gets dropped by the
      //          remote party
      //   mode:  ReadOnly
      //   unit:  1
      //   type:  Trigger
!     //   value: 1
    }
  }
```

4.5.4 Specification Details

Enum `state`

A SAGA stream can be in several states – the complete state diagram is shown in Figure 6. The stream states are:

`New`

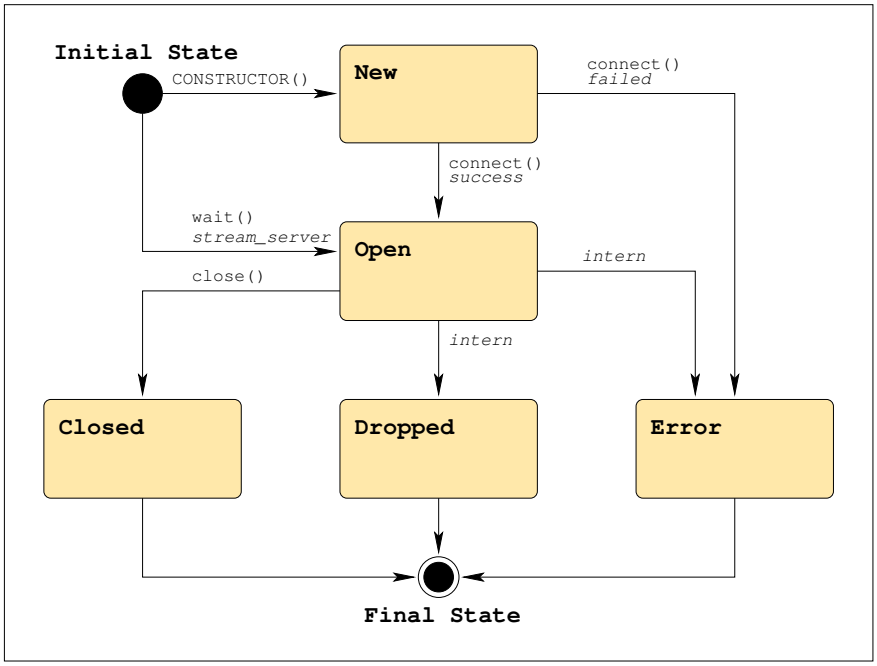

Figure 6: The SAGA stream state model (See Figure 1 for a legend).

A newly constructed stream enters the initial `New` state. It is not connected yet, and no I/O operations can be performed on it. `connect()` must be called to advance the state to `Open` (on success) or `Error` (on failure).

`Open`

The stream is connected to the remote endpoint, and I/O operations can be called. If any error eccurs on the stream, it will move into the `Error` state. If the remote party closes the connection, the stream will move into the `Dropped` state. If `close()` is called on the stream, the stream will enter the `Closed` state.

`Closed`

The `close()` method was called on the stream – I/O is no longer possible. This is a final state.

`Dropped`

The remote party closed the connection – I/O is no longer possible. This is a final state.

`Error`

An error occured on the stream – I/O is no longer possible. This is a final state. The exact reason for reaching this state MUST be available through the `error_handler` interface.

All method calls, apart from the `DESTRUCTOR`, will cause an `IncorrectState` exception if the stream is in a final state.

Enum `activity_type`

The SAGA stream API allows for event driven communication. A stream can flag activities, i.e. `Read`, `Write` and `Exception`, and the application can react on these activities. It is possible to poll for these events (using `wait()` with a potential timeout), or to get asynchronous notification of these events, by using the **respective** metrics.

`Read`

Data are available on the stream, and a subsequent `read()` will succeed.

`Write`

The stream is accepting data, and a subsequent `write()` will succeed.

`Exception`

An error occured on the stream, and a following I/O operation may fail.

Class `stream_service`

The `stream_service` object establishes a listening/server object that waits for client connections. It can *only* be used as a factory for **client** sockets. It doesn't do any read/write I/O.

```
    - CONSTRUCTOR
      Purpose:  create a new stream_service object
!     Format:   CONSTRUCTOR          (in  session   s,
!                                     in  saga::url url = "",
                                      out stream_service obj);
!     Inputs:   s:                    session to be used for
                                      object creation
                url:                  channel name or url,
                                      defines the source side
                                      binding for the stream
```

```
+     InOuts:   -
      Outputs:  obj:                  new stream_service object
+     PreCond:  -
      PostCond: - stream_service can wait for incoming
                  connections.
+               - 'Owner' of name is the id of the context
+                 used to create the stream_service.
+               - the stream_server has 'Exec', 'Query', 'Read'
+                 and 'Write' permissions for '*'.
+     Perms:    -
      Throws:   NotImplemented
                IncorrectURL
+               BadParameter
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
-     Notes:    - if the resource information given in the URL
-                 cannot ever be used by the implementation
-                 (e.g. the hostname is not well formatted,
-                 or the scheme is not available), an
-                 'IncorrectURL' exception is thrown, which
-                 must contain a detailed error message.
      Notes:    - if the given url is an empty string (the
                  default), the implementation will choose an
                  appropriate default value.
+               - the implementation MUST ensure that the given
+                 URL is usable, and a later call to 'serve'
+                 will not fail because of the information given
+                 by the URL - otherwise, a 'BadParameter'
+                 exception MUST be thrown.


    - DESTRUCTOR
      Purpose:  Destructor for stream_service object.
      Format:   DESTRUCTOR           (in stream_service obj)
!     Inputs:   obj:                  object to be destroyed
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
!     PostCond: - the stream_service is closed.
+     Perms:    -
      Throws:   -
      Notes:    - if the instance was not closed before, the
!                 destructor performs a close() on the instance,
```

```
                and all notes to close() apply.

    // inspection
    - get_url
      Purpose:  get URL to be used to connect to this server
!     Format:   get_url              (out saga::url url);
      Inputs:   -
+     InOuts:   -
      Outputs:  url:                  the URL of the connection.
+     PreCond:  -
+     PostCond: -
+     Perms:    -
      Throws:   NotImplemented
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
-     Throws:   -
      Notes:    - returns a URL which can be passed to
!                 the stream constructor to create a connection
                  to this stream_service.


    // stream management
    - serve
      Purpose:  wait for incoming client connections
      Format:   serve                (in  float   timeout,
                                      out stream  client);
      Inputs:   timeout:              number of seconds to wait
!                                     for a client
+     InOuts:   -
      Outputs:  client:               new Connected stream object
+     PreCond:  -
      PostCond: - the returned client is in 'Open' state.
                - the session of the returned client is that of
                  the stream_server.
+     Perms:    - Exec.
+               - Exec for the connecting remote party.
      Throws:   NotImplemented
-               BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
```

```
                NoSuccess
+               Timeout
      Notes:    - if successful, it returns a new stream object
                  that is connected to the client.
-               - returns NULL or equivalent if it times out.
+               - if no client connects within the specified
+                 timeout, a 'Timeout' exception is thrown.
                - if connection setup failed (not on timeout!),
                  the returned client is in the 'Error' state.
                  Its error_handler interface should give
                  detailed information about the reason.
-               - if the resource information given in the URL
-                 (during construction) cannot be used
-                 temporarily (e.g. the port is already taken),
-                 a 'BadParameter' exception is thrown, which
-                 must contain a detailed error message.
-               - in the asynchronous case, the resulting client
-                 stream is passed by reference.  That reference
-                 must be in the 'New' state - otherwise an
-                 'IncorrectState' exception is thrown.
!               - for timeout semantics, see Section 2.


    - close
!     Purpose:  closes a stream service
      Format:   close                (in float timeout)
      Inputs:   timeout               seconds to wait
+     InOuts:   -
      Outputs:  -
!     PreCond:  -
!     PostCond: - no clients are accepted anymore.
+               - no callbacks registered for the
+                 'ClientConnect' metric are invoked.
+     Perms:    -
      Throws:   NotImplemented
                IncorrectState
                NoSuccess
-     Notes:    - if a stream_service was closed earlier
-               an 'IncorrectState' exception is thrown.
+     Notes:    - any subsequent method call on the object
+               MUST raise an 'IncorrectState' exception
+               (apart from DESTRUCTOR and close()).
+               - if close() is implicitely called in the
+               DESTRUCTOR, it will never throw an exception.
+               - close() can be called multiple times, with no
+               side effects.
```

```
-               - it is assumed that a session which opened the
-                 instance can also close it - otherwise the
-                 backend entity must have changed its state,
-                 which causes an 'IncorrectState' exception.
                - for resource deallocation semantics, see
!                 Section 2.
!               - for timeout semantics, see Section 2.
```

Class `stream`

This is the object that encapsulates all client stream objects.

```
    Constructor / Destructor:
    -------------------------

    - CONSTRUCTOR
!     Purpose:  Constructor, initializes a client stream,
!               for later connection to a server.
!     Format:   CONSTRUCTOR          (in  session    s,
!                                     in  saga::url  url,
!                                     out stream  obj);
!     Inputs:   s:                    saga session handle
                url:                  server location as URL
+     InOuts:   -
!     Outputs:  obj:                  new, unconnected stream
                                      instance
+     PreCond:  -
      PostCond: - the state of the socket is 'New'.
+     Perms:    - Query for the stream_service represented by
+               url.
      Throws:   NotImplemented
                IncorrectURL
+               BadParameter
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
!     Notes:    - server location and possibly protocol are
                  described by the input URL - see description
                  above.
```

```
-               - if the resource information given in the URL
-                 cannot ever be used by the implementation
-                 (e.g. the hostname is not well formatted,
-                 or the scheme is not available), an
-                 'IncorrectURL' exception is thrown, which
-                 must contain a detailed error message.
                - the 'url' can be empty (which is the default).
!                 A stream so constructed is only to be used
                  as parameter to an asynchronous
                  stream_server::serve() call.  For such a
                  stream, a later call to connect() will fail.
+               - the implementation MUST ensure that the
+                 information given in the URL are usable -
+                 otherwise a 'BadParameter' exception MUST be
+                 thrown.
                - the socket is only connected after the
                  connect() method is called.


    - DESTRUCTOR
!     Purpose:  destroy a stream object
      Format:   DESTRUCTOR           (in stream obj)
      Inputs:   obj:                  stream to destroy
+     InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - the socket is closed.
+     Perms:    -
      Throws:   -
+     Notes:    - if the instance was not closed before, the
+                 destructor performs a close() on the instance,
+                 and all notes to close() apply.


    Inspection methods:
    -------------------


    - get_url
!     Purpose:  get URL used for creating the stream
!     Format:   get_url               (out saga::url url);
      Inputs:   -
+     InOuts:   -
!     Outputs:  url:                  the URL of the connection.
+     PreCond:  -
+     PostCond: -
+     Perms:    -
```

```
      Throws:   NotImplemented
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
-     Throws:   -
      Notes:    - returns a URL which can be passed to a
                  stream constructor to create another
                  connection to the same stream_service.
                - the returned url may be empty, indicating that
                  this instance has been created with an empty
                  url as parameter to the stream CONSTRUCTOR().


    - get_context
      Purpose:  return remote authorization info
      Format:   get_context           (out context ctx);
      Inputs:   -
+     InOuts:   -
      Outputs:  ctx:                   remote context
      PreCond:  - the stream is, or has been, in the 'Open'
                  state.
!     PostCond: - the returned context is deep copied, and does
                  not share state with any other object.
+     Perms:    -
      Throws:   NotImplemented
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
-     Throws:   -
      Notes:    - the context returned contains the security
                  information from the REMOTE party, and can be
                  used for authorization.
                - if the stream is in a final state, but has
                  been in 'Open' state before, the returned
                  context represents the remote party the stream
!                 has been connected to while it was in 'Open'
                  state.
                - if the stream is not in 'Open' state, and is
                  not in a final state after having been in
                  'Open' state, an 'IncorrectState' exception is
```

```
                thrown.
              - if no security information are available, the
                returned context has the type 'Unknown' and no
!               attributes are attached.
              - the returned context MUST be authenticated, or
                must be of type 'Unknown' as described above.


    Management methods:
    -------------------

    - connect
      Purpose:  Establishes a connection to the target defined
                during the construction of the stream.
      Format:   connect              (void);
      Inputs:   -
+     InOuts:   -
      Outputs:  -
      PreCond:  - the stream is in 'New' state.
      PostCond: - the stream is in 'Open' state.
+     Perms:    Exec for the stream_service represented by the
+               url used for creating this stream instance.
      Throws:   NotImplemented
-               BadParameter
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - on failure, the stream state is changed to
                  'Error'
                - if the stream instance is not in 'New' state,
!                 an 'IncorrectState' exception is thrown.
-               - if the resource information given in the URL
-                 (during construction) cannot be used
-                 temporarily (e.g. the port is already taken),
-                 a 'BadParameter' exception is thrown, which
-                 must contain a detailed error message.


    - close
      Purpose:  closes an active connection
      Format:   close                (in float timeout)
      Inputs:   timeout               seconds to wait
+     InOuts:   -
```

```
      Outputs:  -
!     PreCond:  -
      PostCond: - stream is in 'Closed' state
+     Perms:    -
      Throws:   NotImplemented
                IncorrectState
                NoSuccess
-     Throws:   IncorrectState
-     Notes:    - if a stream was closed earlier (i.e. it is
-                 in 'Closed' or 'Dropped' state), this method
-                 does nothing, and, in particular, does not
-                 throw an 'IncorrectState' exception.
+     Notes:    - any subsequent method call on the object
+                 MUST raise an 'IncorrectState' exception
+                 (apart from DESTRUCTOR and close()).
+               - if close() is implicitely called in the
+                 DESTRUCTOR, it will never throw an exception.
+               - close() can be called multiple times, with no
+                 side effects.
-               - it is assumed that a session which opened the
-                 instance can also close it - otherwise the
-                 backend entity must have changed its state,
-                 which causes an 'IncorrectState' exception.
               - for resource deallocation semantics, see
!                Section 2.
!              - for timeout semantics, see Section 2.


    Stream I/O methods:
    -------------------

    - read
!     Purpose:  Read a data buffer from stream.
-     Format:   read                (inout array<byte> buffer,
+     Format:   read                (inout buffer      buf,
!                                    in    int         len_in = -1,
                                     out   int         len_out);
      Inputs:   len_in:              Maximum number of bytes
!                                    that can be copied into
                                     the buffer.
!     InOuts:   buf:                 buffer to store read data
!                                    into
      Outputs:  len_out:             number of bytes read, if
!                                    successful.
+     PreCond:  - the stream is in 'Open' state.
+     PostCond: - data from the stream are available in the
```

```
+                 buffer.
+     Perms:      Read for the stream_service represented by the
+                 url used for creating this stream instance.
      Throws:     NotImplemented
                  BadParameter
-                 WriteOnly
                  IncorrectState
                  PermissionDenied
                  AuthorizationFailed
                  AuthenticationFailed
                  Timeout
                  NoSuccess
      Notes:      - if the stream is blocking, the call waits
                    until data become available.
                  - if the stream is non-blocking, the call
                    returns immediately, even if no data are
                    available -- that is not an error condition.
!                 - the actually number of bytes read into buffer
!                   is returned in len_out.  It is not an error
!                   to read less bytes than requested, or in fact
!                   zero bytes.
!                 - errors are indicated by returning negative
!                   values for len_out, which correspond to
!                   negatives of the respective ERRNO error code
!                 - the given buffer must be large enough to
!                   store up to len_in bytes, or managed by the
!                   implementation - otherwise a 'BadParameter'
!                   exception is thrown.
+                 - the notes about memory management from the
+                   buffer class apply.
!                 - if len_in is smaller than 0, or not given,
!                   the buffer size is used for len_in.
!                   If that is also not available, a
!                   'BadParameter' exception is thrown.
!                 - if the stream is not in 'Open' state, an
                    'IncorrectState' exception is thrown.
!                 - similar to read (2) as specified by POSIX


    - write
!     Purpose:  Write a data buffer to stream.
-     Format:   write               (in  array<byte>   buffer,
+     Format:   write               (in  buffer        buf,
!                                    in  int           len_in = -1,
                                     out int           len_out);
      Inputs:   len_in:             number of bytes of data in
```

```
                                    the buffer
!               buffer:             buffer containing data
                                    that will be sent out via
                                    socket
+     InOuts:   -
      Outputs:  len_out:            bytes written if successful
      PreCond:  - the stream is in 'Open' state.
+     PostCond: - the buffer data are written to the stream.
+     Perms:    Write for the stream_service represented by the
+               url used for creating this stream instance.
      Throws:   NotImplemented
                BadParameter
-               ReadOnly
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - if the stream is blocking, the call waits
                  until the data can be written.
                - if the stream is non-blocking, the call
                  returns immediately, even if no data are
                  written -- that is not an error condition.
!               - it is not an error to write less than len_in
!                 bytes.
!               - errors are indicated by returning negative
!                 values for len_out, which correspond to
!                 negatives of the respective ERRNO error code
!               - the given buffer must be large enough to
!                 store up to len_in bytes, or managed by the
!                 implementation - otherwise a 'BadParameter'
!                 exception is thrown.
+               - the notes about memory management from the
+                 buffer class apply.
!               - if len_in is smaller than 0, or not given,
!                 the buffer size is used for len_in.
!                 If that is also not available, a
!                 'BadParameter' exception is thrown.
!               - if the stream is not in 'Open' state, an
                  'IncorrectState' exception is thrown.
!               - similar to write (2) as specified by POSIX


    - wait
      Purpose:  check if stream is ready for reading/writing, or
```

```
                if it has entered an error state.
      Format:   wait                 (in  int      what,
                                      in  float    timeout,
                                      out int      cause);
      Inputs:   what:                 activity types to wait for
                timeout:              number of seconds to wait
+     InOuts:   -
      Outputs:  cause:                activity type causing the
                                      call to return
      PreCond:  - the stream is in 'Open' state.
+     PostCond: - the stream can be read from, or written to, or
+               it is in 'Error' state.
+     Perms:    -
      Throws:   NotImplemented
                IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                NoSuccess
-     Throws:   IncorrectState
      Notes:    - wait will only check on the conditions
                  specified by 'what'
                - 'what' is an integer representing
                   OR'ed 'Read', 'Write', or 'Exception' flags.
!               - 'cause' describes the availability of the
                  socket (eg. OR'ed 'Read', 'Write', or
                  'Exception')
!               - for timeout semantics, see Section 2.
!               - if the stream is not in 'Open' state, an
                  'IncorrectState' exception is thrown.
```

4.5.5 Examples

Code Example

```
1   Sample SSL/Secure Client:
2   -------------------------
3
4     Opens a stream connection using native security: the
5     context is passed in implicitly via the default SAGA
6     session's contexts.
7     (GSI or SSL security)
8
9     // C++/JAVA Style
10       ssize_t recvlen;
```

```
      saga::buffer b;
      saga::stream s ("localhost:5000");

      s.connect ();
      s.write   (saga::buffer ("Hello World!"));

      // blocking read, read up to 128 bytes
      recvlen = s.read (b, 128);


   /* C Style */
      ssize_t recvlen;

      SAGA_stream sock  = SAGA_Stream_open ("localhost:5000");
      SAGA_buffer b_in  = SAGA_Buffer_create ("Hello World");
      SAGA_buffer b_out = SAGA_Buffer_create ("Hello World");

      SAGA_Stream_connect (sock);
      SAGA_Stream_write   (sock, b_in);

      /* blocking read, read up to 128 bytes */
      recvlen = SAGA_Stream_read (sock, b_ou, 128);


    c  Fortran Style */
       INTEGER   err,SAGAStrRead,SAGAStrWrite,err
       INTEGER*8 SAGAStrOpen,streamhandle
       CHARACTER buffer(128)
       SAGAStrOpen("localhost:5000",streamhandle)
       call SAGAStrConnect(streamhandle)
       err = SAGAStrWrite(streamhandle,"localhost:5000",12)
       err = SAGAStrRead(streamhandle,buffer,128)


  Sample Secure Server:
  ---------------------

    Once a connection is made, the server can use information
    about the authenticated client to make an authorization
    decision

     // c++ example
        saga::stream_service server ("tcp://localhost/5000");

        saga::stream client;

        // now wait for a connection
        while ( saga::stream::Open != client.get_state () )
        {
          // wait forever for connection
```

```
61          client = server.serve ();
62
63          // get remote security details
64          saga::context ctx = client.get_context ();
65
66          // check if context type is X509, and if DN is the
67          // authorized one
68          if ( ctx.type ()              == "X509"       &&
69               ctx.get_attribute ("DN") == some_auth_dn )
70          {
71            // allowed - keep open and leave loop
72            client.write (saga::buffer ("Hello!"));
73          }
74          else
75          {
76            client.close (); // not allowed
77          }
78        }
79
80        // start activity on client socket...
81
82
83    Example for async stream server
84    -------------------------------
85
86      // c++ example
87      class my_cb : public saga::callback
88      {
89        privat:
90          saga::stream_service ss;
91          saga::stream         s;
92
93        public:
94
95          my_cb (saga::stream_service ss_,
96                 saga::stream         s_)
97          {
98            ss = ss_;
99            s   = s_;
100         }
101
102         bool cb (saga::monitorable mt,
103                  saga::metric      m,
104                  saga::context     c)
105         {
106           s = ss.serve ();
107           return (false); // want to be called only once
108         }
109       }
110
```

```
111     int main ()
112     {
113       saga::stream_service ss;
114       saga::stream         s;
115       my_cb cb (ss, s);
116
117       ss.add_callback ("client_connect", cb);
118
119       while ( true )
120       {
121         if ( s.state != saga::stream::Open )
122         {
123           // no client, yet
124           sleep (1);
125         }
126         else
127         {
128           // handle open socket
129           s.write ("Hello Client\r\n", 14);
130           s.close ();
131
132           // restart listening
133           ss.add_callback ("client_connect", cb);
134         }
135       }
136
137       return (-1); // unreachable
138     }
```

4.6 SAGA Remote Procedure Call

GridRPC is one of the few high level APIs that have been specified by the GGF [19]. Thus including the GridRPC specification in the SAGA API benefits both SAGA and the GridRPC effort: SAGA becomes more complete and provides a better coverage of its use cases with a single **Look-&-Feel**, whilst GridRPC gets embedded into a set of other tools of similar scope, which opens it to a potentially wider user community, and ensures its further development.

Semantically, the methods defined in the GridRPC specification, as described in GFD.52 [19], 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.

The rpc class constructor initialises the remote function handle. This process may involve connection setup, service discovery, etc. The rpc class further offers one method 'call', which invokes the remote procedure, and returns the respective return data and values. The asynchronous call versions described in the GridRPC specification are realised by the SAGA task model, and are not represented as separate calls here.

In the constructor, the remote procedure to be invoked is specified by a URL, with the syntax:

```
gridrpc://server.net:1234/my_function
```

with the elements responding to:

`gridrpc`	–	scheme	–	identifying a grid rpc operation
`server.net`	–	server	–	server host serving the rpc call
`1234`	–	port	–	contact point for the server
`my_function`	–	name	–	name of the remote method to invoke

All elements can be empty, which allows the implementation to fall back to a default remote method to invoke.

The argument and return value handling is very basic, and reflects the traditional scheme for remote procedure calls, that is, an array of structures acts as variable parameter vector. For each element of the vector, the `parameter` struct describes its data `buffer`, the `size` of that buffer, and its input/output `mode`.

The `mode` value has to be initialized for each `parameter`, and `size` and `buffer` values have to be initialized for each `In` and `InOut` struct. For `Out` parameters, `size` may have the value `0` in which case the `buffer` must be un-allocated, and

is to be created (e.g. allocated) by the SAGA implementation upon arrival of the result data, with a size sufficient to hold all result data. The `size` value is to be set by the implementation to the allocated buffer size. SAGA language bindings MUST prescribe the responsibilities for releasing the allocated buffer, according to usual procedures in the respective languages.

When an `Out` or `InOut` struct uses a pre-allocated buffer, any data exceeding the buffer size are discarded. The application is responsible for specifying correct buffer sizes for pre-allocated buffers; otherwise the behaviour is undefined.

This argument handling scheme allows efficient (copy-free) passing of parameters. The parameter vector must be passed by reference because it is specified as `inout` in SIDL. (See also Section 2.2.)

4.6.1 RPC Permissions

The SAGA API allows for application level authorization of RPC calls an application is able to set permissions on ***`saga::rpc`*** *instances. Not all implementations will be able to fully implement that security model – the implementation MUST carefully document which permissions are supported, and which are not.*

4.6.2 Specification

```
  package saga.rpc
  {
    enum io_mode
    {
      In    = 1,          // input  parameter
      Out   = 2,          // output parameter
      InOut = 3           // input and output parameter
    }

-   struct parameter
-   {
-     int         size;   // number of bytes in buffer
-     array<byte> buffer; // data
-     io_mode     mode;   // parameter mode
-   }
-
+   class parameter : extends saga::buffer
+             // from buffer saga::object
+             // from object saga::error_handler
```

```
+   {
+     CONSTRUCTOR (in    array<byte>       data = "",
+                  in    int               size = 0,
+                  in    io_mode           mode = In,
+                  out   buffer            obj);
+
+     set_io_mode (in    io_mode           mode);
+     get_io_mode (out   io_mode           mode);
    }

    class rpc : implements   saga::object
                implements   saga::async
+               implements   saga::permissions
              // from object  saga::error_handler
    {
!     CONSTRUCTOR (in    session           s,
!                  in    saga::url         url = "",
!                  out   rpc               obj          );
      DESTRUCTOR  (in    rpc               obj          );

      // rpc method invocation
      call        (inout array<parameter>  parameters   );

      // handle management
      close       (in    float             timeout = 0.0);
    }
  }
```

4.6.3 Specification Details

Enum `io_mode`

The ***io_mode*** *enum specifies the modus of the* ***rpc::parameter*** *instances:*

`In`

The parameter is an input parameter: its initial value will be evaluated, and its data buffer will not be changed during the invocation of ***call()****.*

`Out`

The parameter is an output parameter: its initial value will not be evaluated, and its data buffer will likely be changed during the invocation of ***call()****.*

`InOut`

The parameter is input and output parameter: its initial value will not evaluated, and its data buffer will likely be changed during the invocation of `call()`.

Class `parameter`

The `parameter` class inherits the `saga::buffer` class, and adds one additional state attribute: `io_mode`, which is read-only. With that addition, the new class can conveniently be used to define input, inout and output parameters for RPC calls.

```
+   - CONSTRUCTOR
+     Purpose:  create an parameter instance
+     Format:   CONSTRUCTOR          (in  array<byte> data = "",
+                                     in  int         size = -1,
+                                     in  io_mode     mode = In,
+                                     out parameter       obj);
+     Inputs:   type:                 data to be used
+               size:                 size of data to be used
+               io_mode:              type of parameter
+     InOuts:   -
+     Outputs:  parameter:            the newly created parameter
+     PreCond:  -
+     PostCond: -
+     Perms:    -
+     Throws:   NotImplemented
+               BadParameter
+               NoSuccess
+     Notes:    - all notes from the buffer CONSTRUCTOR apply.
+
+
+   - DESTRUCTOR
+     Purpose:  destroy an parameter instance
+     Format:   DESTRUCTOR           (in  parameter obj);
+     Inputs:   obj:                  the parameter to destroy
+     InOuts:   -
+     Outputs:  -
+     PreCond:  -
+     PostCond: -
+     Perms:    -
+     Throws:   -
```

```
+     Notes:    - all notes from the buffer DESTRUCTOR apply.
+
+
+   - set_io_mode
+     Purpose:  set io_mode
+     Format:   set_io_mode          (in  io_mode mode);
+     Inputs:   mode:                 value for io mode
+     InOuts:   -
+     Outputs:  -
+     PreCond:  -
+     PostCond: -
+     Perms:    -
+     Throws:   -
+     Notes:    -
+
+   - get_io_mode
+     Purpose:  retrieve the current value for io mode
+     Format:   get_io_mode          (out io_mode mode);
+     Inputs:   -
+     InOuts:   -
+     Outputs:  mode:                 value of io mode
+     PreCond:  -
+     PostCond: -
+     Perms:    -
+     Throws:   -
+     Notes:    -
```

Class `rpc`

This class represents a remote function handle, which can be called (repeatedly), and returns the result of the respective remote procedure invocation.

```
-   Constructor / Destructor:
-   -------------------------
-
    - CONSTRUCTOR
!     Purpose:  initializes a remote function handle
!     Format:   CONSTRUCTOR  (in  session   s,
!                             in  saga::url url = "",
                              out rpc       obj);
!     Inputs:   s:            saga session to use
                funcname:     name of remote method to
```

```
                              initialize
      InOuts:   -
      Outputs:  obj           the newly created object
+     PreCond:  -
+     PostCond: - the instance is open.
+     Perms:    Query
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
                DoesNotExist
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
-     Notes:    - if the given 'url' cannot ever be used by
-               - the implementation (e.g. the hostname is
-               - not well formatted, or the scheme is not
-               - available), an 'IncorrectURL' exception is
-               - thrown, which must contain a detailed error
-               - message.
!     Notes:    - if url is not given, or is empty (the
!               - default), the implementation will choose an
                - appropriate default value.
                - according to the GridRPC specification, the
                - constructor may or may not contact the RPC
                - server; absence of an exception does not imply
                - that following RPC calls will succeed, or that
                - a remote function handle is in fact available.
                - the following mapping MUST be applied from
                - GridRPC errors to SAGA exceptions:
                - GRPC_SERVER_NOT_FOUND   : BadParameter
                - GRPC_FUNCTION_NOT_FOUND : DoesNotExist
                - GRPC_RPC_REFUSED        : AuthorizationFailed
                - GRPC_OTHER_ERROR_CODE   : NoSuccess
                - non-GridRPC based implementations SHOULD ensure
!               - upon object construction that the remote handle
                - is available, for consistency with the
                - semantics on other SAGA object constructors.

    - DESTRUCTOR
      Purpose:  destroy the object
      Format:   DESTRUCTOR           (in  rpc  obj)
      Inputs:   obj:                  the object to destroy
      InOuts:   -
      Outputs:  -
```

```
+     PreCond:  -
+     PostCond: - the instance is closed.
+     Perms:    -
      Throws:   -
      Notes:    - if the instance was not closed before, the
!                 destructor performs a close() on the instance,
                  and all notes to close() apply.


    - call
      Purpose:  call the remote procedure
      Format:   call         (inout array<parameter> param);
      Inputs:   -
      In/Out:   param:        argument/result values for call
      InOuts:   -
      Outputs:  -
+     PreCond:  - the instance is open.
+     PostCond: - the instance is avaiable for another call()
+                 invocation, even if the present call did not
+                 yet finish, in the asynchronous case.
+     Perms:    Exec
      Throws:   NotImplemented
                IncorrectURL
                BadParameter
                DoesNotExist
+               IncorrectState
                PermissionDenied
                AuthorizationFailed
                AuthenticationFailed
                Timeout
                NoSuccess
      Notes:    - according to the GridRPC specification, the
                  RPC server might not be contacted before
                  invoking call(). For this reason, all notes to
                  the object constructor apply to the call()
                  method as well.
                - if an implementation finds inconsistent
!                 information in the parameter vector, a
                  'BadParameter' exception is thrown.
                - arbitrary backend failures (e.g. semantic
                  failures in the provided parameter stack, or
                  any errors occuring during the execution of
                  the remote procedure) MUST be mapped to a
!                 'NoSuccess' exception, with a descriptive
                  error message.  That way, error semantics of
                  the SAGA implementation and of the RPC
```

```
                  function implementation are strictly
                  distinguished.
+                 - the notes about memory management from the
+                   buffer class apply.


    - close
      Purpose:  closes the rpc handle instance
      Format:   close              (in  float timeout = 0.0);
      Inputs:   timeout             seconds to wait
      InOuts:   -
      Outputs:  -
+     PreCond:  -
+     PostCond: - the instance is closed.
+     Perms:    -
      Throws:   NotImplemented
                IncorrectState
                NoSuccess
-     Notes:    - an 'IncorrectState' exception is thrown if the
-                 object was closed before.
+     Notes:    - any subsequent method call on the object
+                 MUST raise an 'IncorrectState' exception
+                 (apart from DESTRUCTOR and close()).
+               - if close() is implicitely called in the
+                 DESTRUCTOR, it will never throw an exception.
+               - close() can be called multiple times, with no
+                 side effects.
-               - it is assumed that a session which opened the
-                 instance can also close it - otherwise the
-                 backend entity must have changed its state,
-                 which causes an 'IncorrectState' exception.
               - for resource deallocation semantics, see
!                Section 2.
!              - for timeout semantics, see Section 2.
```

4.6.4 Examples

Code Example

```
1  // c++ example
2  // call a remote matrix multiplication A = A * B
3  try
4  {
5    rpc rpc ("gridrpc://rpc.matrix.net/matrix-mult");
6
```

```
7     std::vector <saga::rpc::parameter> params (2);
8
9     params[0].set_data (A); // ptr to matrix A
10    params[0].set_io_mode (saga::rpc::InOut);
11
12    params[1].set_data (B); // ptr to matrix B
13    params[1].set_io_mode (saga::rpc::In);
14
15    rpc.call (params);
16
17    // A now contains the result
18  }
19  catch ( const saga::exception & e)
20  {
21    std::err << "SAGA error: "
22             << e.get_message ()
23             << std::endl;
24  }
25
26  +-------------------------------------------------------------+
27
28  // c++ example
29  // call a remote matrix multiplication C = A * B
30  try
31  {
32    rpc rpc ("gridrpc://rpc.matrix.net//matrix-mult-2");
33
34    std::vector <saga::rpc::parameter> params (3);
35
36    params[0].set_data (NULL); // buffer will be created
37    params[0].set_io_mode (saga::rpc::Out);
38
39    params[1].set_data (A); // ptr to matrix A
40    params[1].set_io_mode (saga::rpc::In);
41
42    params[2].set_data (B); // ptr to matrix B
43    params[2].set_io_mode (saga::rpc::In);
44
45    rpc.call (params);
46
47    // params[0].get_data () now contains the result
48  }
49  catch ( const saga::exception & e)
50  {
51    std::err << "SAGA error: "
52             << e.get_message ()
53             << std::endl;
54  }
55
56  +-------------------------------------------------------------+
```

```
57
58  // c++ example
59  // asynchronous version of A = A * B
60  try
61  {
62    rpc rpc ("gridrpc://rpc.matrix.net/matrix-mult");
63
64    std::vector <saga::rpc::parameter> params (2);
65
66    params[0].set_data (A); // ptr to matrix A
67    params[0].set_io_mode (saga::rpc::InOut);
68
69    params[1].set_data (B); // ptr to matrix B
70    params[1].set_io_mode (saga::rpc::In);
71
72    saga::task t = rpc.call <saga::task::ASync> (params);
73
74    // do something else
75
76    t.wait ();
77    // A now contains the result
78  }
79  catch ( const saga::exception & e)
80  {
81    std::err << "SAGA error: "
82             << e.get_message ()
83             << std::endl;
84  }
85
86  +-------------------------------------------------------------+
87
88  // c++ example
89  // parameter sweep example from
90  // http://ninf.apgrid.org/documents/ng4-manual/examples.html
91  //
92  // Monte Carlo computation of PI
93  //
94  try
95  {
96    saga::url     uri[NUM_HOSTS]; // initialize...
97    long times, count[NUM_HOSTS], sum;
98
99    std::vector <saga::rpc> servers;
100
101   // create the rpc handles for all URIs
102   for ( int i = 0; i < NUM_HOSTS; ++i )
103   {
104     servers.push_back (saga::rpc (uri[i]));
105   }
106
```

```
107     // create persistent storage for tasks and parameter structs
108     saga::task_container tc;
109     std::vector <std::vector <saga:parameter> > params;
110
111     // fill parameter structs and start async rpc calls
112     for ( int i = 0; i < NUM_HOSTS; ++i )
113     {
114       std::vector <saga::rpc::parameter> param (3);
115
116       param[0].set_data (i); // use as random seed
117       param[0].set_io_mode (saga::rpc::In);
118
119       param[1].set_data (times);
120       param[1].set_io_mode (saga::rpc::In);
121
122       param[2].set_data (count[i]);
123       param[2].set_io_mode (saga::rpc::Out);
124
125       // start the async calls
126       saga::task t = servers[i].call <saga::task::Async> (param);
127
128       // save the task;
129       tc.add (t[i]);
130
131       // save the parameter structs
132       params.push_back (param);
133     }
134
135     // wait for all async calls to finish
136     tc.wait (saga::task::All);
137
138     // compute and print pi
139     for ( int i = 0; i < NUM_HOSTS; ++i )
140     {
141       sum += count[i];
142     }
143
144     std::out << "PI = "
145              << 4.0 * ( sum / ((double) times * NUM_HOSTS))
146              << std::endl;
147   }
148   catch ( const saga::exception & e)
149   {
150     std::err << "SAGA error: "
151              << e.get_message ()
152              << std::endl;
153   }
```

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.

Tom Goodale
t.r.goodale@cs.cardiff.ac.uk
Cardiff School of Computer Science
5, The Parade, Roath
Cardiff, CF24 3AA
United Kingdom

Shantenu Jha
s.jha@ucl.ac.uk
Centre for Computational Science
University College London
London, WC1H 0AJ
United Kingdom

Hartmut Kaiser
hkaiser@cct.lsu.edu
Center for Computation and Technology
Louisiana State University
216 Johnston Hall
70803 Baton Rouge
Louisiana, USA

Thilo Kielmann
kielmann@cs.vu.nl
Vrije Universiteit
Dept. of Computer Science
De Boelelaan 1083
1081HV Amsterdam
The Netherlands

Pascal Kleijer
k-pasukaru@ap.jp.nec.com
NEC Corporation
HPC Marketing Promotion
1-10, Nisshin-cho, Fuchu
183-8501 Tokyo
Japan

Andre Merzky
andre@merzky.net
VU (see Kielmann)
CCT/LSU (see Kaiser)

John Shalf
jshalf@lbl.gov
Lawrence Berkeley
National Laboratory
Mailstop 50F
1 Cyclotron Road
94720 Berkeley
California, USA

Christopher Smith
csmith@platform.com
Platform Computing Inc.
USA

The initial version of the presented SAGA API was drafted by the SAGA 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:

Hrabri Rajic (Intel), Keith Jackson (LBL), David Konerding (LBL), Gregor von Laszewski (ANL).

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:

Gabriele Allen (LSU), Stephan Hirmer (LSU), Craig Lee (Aerospace Corporation), Hidemoto Nakada (AIST), Steven Newhouse (OMII-UK), Stephen Pickles (University of Manchester), Ed Seidel (LSU), Derek Simmel (PSC), Yusuke Tanimura (AIST), Osamu Tatebe (University of Tsukuba).

5.2 Intellectual Property Statement

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

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

5.3 Disclaimer

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

5.4 Full Copyright Notice

Copyright (C) Open Grid Forum (2006). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the OGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the OGF Document process must be followed, or as required to translate it into languages other than English.

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

Appendix

A SAGA Code Examples

This appendix shows a couple of SAGA examples in different languages. As stated in the introduction, these examples are not normative – language bindings are outside the scope of this document. This appendix is rather supposed to illustrate how the authors imagine the use of the API in various languages.

We hope that the examples illustrate that the API stays SIMPLE in various language incarnations, as was the major design intent for the _**S**_AGA API.

Code Example

```

Example 1 (C++): Object State:
==============================

  // This example illustrates the expected life
  // times of object states.  State is shared in
  // these cases, as only shallow copies occur.

  int main (void)
  {
    { // task scope
      saga::task t;

      { // file scope
        saga::file f;

        { // session scope
          saga::session s;

          { // context scope
            saga::context c (saga::context::UserPass);

            s.add_context (c);
            f (s, saga::url ("file:////tmp/data.bin"));
            t = f.copy <saga::task::Task>
                  (saga::url ("file:////tmp/data.bak"));

          } // leave context scope
            // session keeps context state

        } // leave session scope
          // file keeps session state

      } // file scope
```

```
35          // task keeps file state
36
37        t.run ();
38        // task runs, and uses state of file, session,
39        // and context.
40        t.wait ();
41
42      } // task scope
43        // task    releases file state
44        // file    releases session state
45        // session releases context state
46
47      return (0);
48    }
49
50
51  +-------------------------------------------------------------+
52
53  Example 2: Files:
54  =================
55
56    open a file. if its size is > 10, then read the first 10
57    bytes into a string, print it, end return it.
58
59    ---------------------------------------------------------------
60    Example 2a: C++
61    ---------------------------------------------------------------
62    // c++ example
63    void head (const saga::url url)
64    {
65      try {
66        // get type and other infos
67        saga::file f (url);
68
69        off_t size = f.get_size ();
70
71        if ( size > 10 )
72        {
73          char   buf[11];
74
75          ssize_t len_out = f.read (saga::buffer (buf));
76
77          if ( 10 == len_out )
78          {
79            std::cout << "head: "
80                      << buffer.get_data ()
81                      << std::endl;
82          }
83        }
84      }
```

```
85
86        catch ( const saga::exception & e )
87        {
88          std::cerr << "Oops! SAGA error: "
89                    << e.get_message ()
90                    << std::endl;
91        }
92
93        return;
94      }
95      ---------------------------------------------------------------
96      ---------------------------------------------------------------
97      Example 2b: C
98      -------------
99        void head (const SAGA_URL url)
100       {
101         SAGA_File my_file = SAGA_File_create (url);
102
103         if ( NULL == my_file )
104         {
105           fprintf (stderr, "Could not create SAGA_File "
106                            "for %s: %s\n",
107                    SAGA_URL_get_url (url),
108                    SAGA_Session_get_error (theSession));
109           return (NULL);
110         }
111
112         off_t size = SAGA_File_get_size (my_file);
113
114         if ( size < 0 )
115         {
116           fprintf (stderr, "Could not determine file size "
117                            "for %s: %s\n",
118                    SAGA_URL_get_url (url),
119                    SAGA_Session_get_error (theSession));
120           return (NULL);
121         }
122         else if ( size >= 10 )
123         {
124           SAGA_buffer b = SAGA_Buffer_create ();
125           size_t bufflen;
126
127           ssize_t ret = SAGA_File_read (my_file, b, 10);
128
129           if ( ret < 0 )
130           {
131             fprintf (stderr, "Could not read file %s: %s\n",
132                      SAGA_URL_get_url (url),
133                      SAGA_Session_get_error (theSession));
134           }
```

```
135           else if ( ret < 10 )
136           {
137             fprintf (stderr, "head: short read: %d\n", ret);
138           }
139           else
140           {
141             printf ("head: '%s'\n", SAGA_Buffer_get_data (b));
142           }
143         }
144         else
145         {
146           fprintf (stderr, "head: file %s is too short: %d\n",
147                    file, size);
148         }
149
150         return;
151       }
152
153     ----------------------------------------------------------------
154     Example 2c: Java
155     ----------------
156
157     import org.ogf.saga.URI;
158     import org.ogf.saga.buffer.Buffer;
159     import org.ogf.saga.buffer.BufferFactory;
160     import org.ogf.saga.file.File;
161     import org.ogf.saga.file.FileFactory;
162     import org.ogf.saga.namespace.Flags;
163     import org.ogf.saga.session.Session;
164
165     public class Example {
166       // open a file. if its size is >= 10, then read the first
167       // 10 bytes into a string, print it, end return it.
168       public String head(Session session, URI uri)
169       {
170         try
171         {
172           File f = FileFactory.createFile(session, uri, Flags.READ);
173           long size = f.getSize();
174
175           if (10 <= size) {
176             Buffer    buffer = BufferFactory.createBuffer(10);
177             int       res    = f.read(10, buffer);
178
179             if (10 == res) {
180               System.out.println("head: " + buffer);
181             } else {
182               System.err.println("head: read is short! " + res);
183             }
184             return new String(buffer.getData());
```

```
185           } else {
186             System.err.println("file is too small: " + size);
187           }
188         } catch (Exception e) {
189           // catch any possible error - see elsewhere for better
190           // examples of error handling in SAGA
191           System.err.println ("Oops! " + e);
192         }
193
194         return null;
195       }
196     }
197     ---------------------------------------------------------------
198     Example 2d: Perl ('normal' error handling)
199     ------------------------------------------
200
201       sub head ($)
202       {
203         my $url     = shift;
204         my $my_file = new saga::file (url)
205                   or die ("can't create file for $url: $!\n");
206
207         my $size    = my_file->get_size ();
208
209         if ( $size > 10 )
210         {
211           my $buffer = new saga::buffer (10)l
212           my $ret    = my_file->read ($buffer)
213                   or die ("can't read from file $url: $!\n");
214
215           if ( $ret == 10 )
216           {
217             print "head: ", $buffer->get_data (), "\n";
218           }
219           else
220           {
221             printf STDERR "head: short read: %d\n" ($buffer);
222           }
223         }
224         else
225         {
226           print STDERR "file $url is too short: $size\n";
227         }
228
229         return;
230       }
231
232     ---------------------------------------------------------------
233     Example 2e: Perl (exceptions)
234     -----------------------------
```

```
235     sub head ($)
236     {
237       my $url     = shift;
238
239       eval
240       {
241         my $my_file = new saga::file (url);
242         my $size    = my_file->get_size ();
243
244         if ( $size > 10 )
245         {
246           my $buffer = new saga::buffer (10)l
247           my $ret    = my_file->read ($buffer);
248
249           if ( $ret == 10 )
250           {
251             print "head: ", $buffer->get_data (), "\n";
252           }
253           else
254           {
255             printf "head: short read: %d \n", length ($buffer);
256           }
257         }
258         else
259         {
260           print "file $url is too short: $size\n";
261         }
262       }
263
264       if ( $@ =~ /^saga/i )
265       {
266         print "catched saga error: $@\n" if $@;
267       }
268
269       return;
270     }
271
272   ----------------------------------------------------------------
273   Example 2f: Fortran 90
274   ----------------------
275
276   C Fortran 90 example
277      SUBROUTINE HEAD(session, url, buffer)
278
279      INTEGER      :: session, url, file, size, bufflen
280      CHARACTER*10 :: buffer
281
282      CALL SAGA_FILE_CREATE(session, url, file)
283      CALL SAGA_FILE_GET_SIZE(file, size)
284
```

```
      IF size .GT. 10 THEN

        CALL SAGA_FILE_READ(file, 10, buffer, bufflen)

        IF bufflen .EQ. 10 THEN
          WRITE(5, *) 'head: ', buffer
        ELSE
          WRITE(5, *) 'head: short read: ', bufflen
        ENDIF
      ELSE
        WRITE(5, *) 'file is too short'
      ENDIF

      END

  ------------------------------------------------------------
  Example 2g: Python
  ------------------
  # Python example
  def head (session,url):

    try:
      my_file = saga.file(session,url)
      size = my_file.get_size()

      if (size > 10):
        my_buffer = saga.buffer (10)
        ret = my_file.read (my_buffer)
        if (ret == 10):
          print "head: ", my_buffer.get_data ()
        else
          print "head: short read: ", ret
      else
        print "head: file is too short: ", size

    # catch any possible error - see elsewhere for better
    # examples of error handling in SAGA
    except saga.Exception, e:
      print "Oops! SAGA error: ", e.get_message ()
```

References

[1] W. Allcock, I. Foster, and R. Madduri. Reliable Data Transport: A Critical Service for the Grid. Technical report, Global Grid Forum 11, June 2004.

[2] G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser, T. Kielmann, A. Merzky, R. van Nieuwpoort, A. Reinefeld, F. Schintke, T. Schütt, E. Seidel, and B. Ullmer. The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid. *Proceedings of the IEEE*, 93(3):534–550, 2005.

[3] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva. Job Submission Description Language (JSDL) Specification V1.0. Grid Forum Document GFD.56, 2005. Global Grid Forum.

[4] Babel Project. Scientific Interface Definition Language (SIDL). `http://www.llnl.gov/CASC/components/babel.html` .

[5] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Standard), Jan. 2005.

[6] 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`.

[7] M. Drescher and A. Anjomshoaa. JSDL Parameter Sweep Job Extension (draft 006). Grid Forum Working Draft, 2007. Open Grid Forum.

[8] M. Drescher and A. Anjomshoaa. JSDL SPMD Application Extension, Version 1.0 (draft 008). Grid Forum Working Draft, 2007. Open Grid Forum.

[9] DRMAA Working Group. Open Grid Forum. `http://forge.ogf.org/sf/projects/drmaa-wg/`.

[10] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. V. Reich. The Open Grid Services Architecture, Version 1.0. Technical report, Global Grid Forum, 2005. GFD.30.

[11] Grid Checkpoint and Recovery Working Group (GridCPR), Open Grid Forum (OGF). `http://forge.ogf.org/sf/projects/gridcpr-wg`.

[12] A. Grimshaw, S. Newhouse, D. Pulsipher, and M. Morgan. OGSA Basic Execution Service, Version 1.0. Working document, OGSA Basic Execution Service Working Group, Open Grid Forum, September 2006. `http://www.ogf.org/pipermail/ogsa-bes-wg/attachments/20060906/c1849ef3/attachment-0003.doc`.

[13] S. Hirmer, H. Kaiser, A. Merzky, A. Hutanu, and G. Allen. Generic Support for Bulk Operations in Grid Applications. In *MCG '06: Proceedings of the 4th International Workshop on Middleware for Grid Computing*, page 9, New York, NY, USA, November 2006. ACM Press.

[14] F. Isaila and W. Tichy. Clusterfile: A flexible physical layout parallel file system. *Concurrency and Computation: Practice and Experience*, 15(7–8):653–679, 2003.

[15] JSDL Working Group. Open Grid Forum. `http://forge.ogf.org/sf/projects/jsdl-wg/`.

[16] 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` .

[17] A. Merzky and S. Jha. A Collection of Use Cases for a Simple API for Grid Applications. Grid Forum Document GFD.70, 2006. Global Grid Forum.

[18] A. Merzky and S. Jha. A Requirements Analysis for a Simple API for Grid Applications. Grid Forum Document GFD.71, 2006. Global Grid Forum.

[19] 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.

[20] M. Pereira, O. Tatebe, L. Luan, and T. Anderson. Resource Namespace Service Specification. Working document, Grid File Systems Working Group, Open Grid Forum, September 2006. `http://www.ogf.org/pipermail/gfs-wg/attachments/20060922/f2e549ed/attachment-0001.pdf`.

[21] *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.

[22] *Portable Operating System Interface (POSIX) – Part 2: Shell and Utilities (Volume 1)*. Information technology – Portable Operating System Interface (POSIX). IEEE Computer Society, 345 E. 47th St, New York, NY 10017, USA, 1993.

[23] *Portable Operating System Interface (POSIX) – Part 2: Shell and Utilities (Volume 2)*. Information technology – Portable Operating System Interface (POSIX). IEEE Computer Society, 345 E. 47th St, New York, NY 10017, USA, 1993.

[24] H. Rajic, R. Brobst, W. Chan, F. Ferstl, J. Gardiner, J. P. Robarts, A. Haas, B. Nitzberg, H. Rajic, and J. Tollefsrud. Distributed Resource Management Application API Specification 1.0. Grid Forum Document GFD.22, 2004. Global Grid Forum.